Theoretical Computer Science 101 (1992) 289-335
Elsevier

A compositional axiomatization of Statecharts*

J.J.M. Hooman

Department of Mathematics and Computing Science, Eindhoven University of Technology, 5600 MB Eindhoven, The Netherlands

S. Ramesh**

Department of Computer Science and Engineering, Indian Institute of Technology, Bombay 400 076, India

W.P. de Roever

Institut für Informatik und Praktische Mathematik II, Christian-Albrechts-Universität zu Kiel, 2300 Kiel 1, Germany

Abstract

Hooman, J.J.M., S. Ramesh and W.P. de Roever, A compositional Axiomatization of Statecharts, Theoretical Computer Science 101 (1992) 289-335.

Statecharts is a behavioural specification language proposed for specifying large real-time, event-driven, reactive systems. It is a graphical language based on state-transition diagrams for finite state machines extended with many features like hierarchy, concurrency, broadcast communication and time-out. We supply Statecharts with a compositional axiomatization for both safety and liveness properties. By generating external events symbolically, Statecharts can be executed, thereby turning it into a programming language for real-time concurrency (as well as enabling rapid prototyping). As such it is well suited for compositional program verification. In addition to our compositional axiomatic system, we give a denotational semantics and prove that the axiomatization is sound and relatively complete with respect to this semantics.

1. Introduction

This paper deals with formal specification and verification of real-time reactive systems. Reactive systems [14] are typically event-driven, often have critical time requirements, and, most importantly, they are continuously interacting with their environment. Typical examples are telecommunication networks and avionic systems. Formal specification of real-time reactive systems is an important area of

* This research was supported by Esprit Project 937 (DESCARTES) and Esprit-BRA project 3096 (SPEC).

** The work described here was done while this author was at the Eindhoven University of Technology, partially supported by the Netherlands National Facility for Informatics (NFI).

research judging by the sheer number of proposed specification languages such as Statecharts [11], Esterel [2], Lustre [3] and Signal [9]. All these specification languages are based on operational descriptions that characterize how a system evolves. As such they are perfectly suited for simulation purposes to analyze and to debug the description of the system under development. To achieve additional confidence in the specification of the intended system, our aim is to complement behavioural specifications with more abstract property-based specifications. More precisely, we give a logical specification language and develop a sound and relatively complete axiomatic system to verify a behavioural specification with respect to a property-based specification.

In this paper we axiomatize the behavioural specification language Statecharts [11]. This visual formalism [12] can be considered as an extension of conventional state-transition diagrams of finite state machines with hierarchy, concurrency and a communication mechanism. A specification written in the language Statecharts is called a statechart. Originally, this language has been designed by Harel to support the development of a complex avionics system. Currently the formalism is indeed used in aircraft industry, but it has been applied successfully in other industrial development projects as well. See also [6], where it is argued that Statecharts can be beneficially used as a hardware description language. Statecharts is used in a computerized graphical tool, called STATEMATE [13], to represent the behavioural view of the system under development. STATEMATE supports two other views: the functional view (describing dataflow and activities) and the structural view (describing the static structure of physical modules and channels). This tool allows us to simulate the specified system and it incorporates several analysis capabilities, most of them based on finite state methods. The full Statecharts language, however, allows the use of variables and hence completely automated verification is impossible. Furthermore, the limits of automated finite state verification are almost reached (see, e.g. [32]).

Our aim is to develop an axiomatization of Statecharts which is compositional, that is, properties of a compound statechart should be derived purely from properties of its constituents without referring to the internal structure of these constituents. Compositionality requires syntactic operators for building large statecharts from smaller ones. Our axiomatization is based on a textual syntax for Statecharts which has been proposed in [20]. Statecharts are related to our logical specification language by formulae of the form S **sat** φ, meaning that statechart S satisfies assertion φ. Assertions are written in a first-order typed language which is strong enough to express safety and liveness properties. Safety properties are properties that can be falsified in finite time, such as, "event e is generated within five steps". A typical example of a liveness property is "eventually event e is generated".

In the literature many axiomatic systems are presented (see [1, 16, 18, 19, 27, 31, 34], to mention a few) for deducing properties of real-time or reactive systems. All these systems deal with CSP-like languages [16, 23] which lack many of the features of Statecharts, such as interrupts and broadcast communication. Often 'sat'-based

formalisms are used for specifications [1, 16, 19, 27, 34]. Some of the works [16, 17, 34] deal with safety properties while others [1, 27, 19, 31] derive both safety and liveness properties. For expressing liveness properties the latter works make use of temporal logic, and compositionality is achieved in [1, 19, 27] by using chop operators or greatest fixed point operators in the assertion language. In this paper such operators are avoided by making explicit reference to time and using the availability of unbounded time. As Lamport already observed in [24], this simplifies the step from real-time safety to liveness.

In [34, 35] it has been shown how the kind of sat-system as developed here can be turned into a (compositional) generalized Hoare system based on pre- and post-conditions on computation histories, and into trace-invariant systems in which concurrency is characterized by a communication interface. The remaining prevalent styles of compositional specification and proof systems, viz. the assumption-commitment paradigm for distributed computation (tracing back to early work of Misra and Chandy [26]) as well as the rely-guarantee paradigm (tracing back to early work of Jones [22]), can be canonically converted into trace-invariant proof systems as indicated in [36]. Once future research has determined which adaptation of these three specification styles—generalized Hoare logic, trace-invariant, assumption-commitment—is most natural to reason about Statecharts, our proof system can be lifted to suit that style using the canonical proof-transformation techniques developed by Zwiers et al., cited above. This provides added motivation to our aim to develop first a compositional axiomatization for Statecharts which is as simple as possible.

To prove soundness and relative completeness of our axiomatic system, we have defined a denotational, and hence compositional, semantics for Statecharts. An early operational, noncompositional, semantics for Statecharts can be found in [15]. In [20] a compositional semantic model with minimal amount of nonobservable entities (i.e. a fully abstract semantics) has been presented. The denotations of this semantics are prefix-closed sets of linear histories; infinite computations are represented by all their finite prefixes. Our semantics is derived from this model, but since it has to serve as a basis for our formalism expressing liveness properties, several changes have been made. Instead of using prefix closed sets, our histories represent complete (possibly infinite) computations. Furthermore, in [20] the least fixed point is used to describe a looping construct whereas we use the greatest fixed point to obtain also infinite computations. A discussion about choices in defining the semantics of Statecharts can be found in [21]. In [8] a related denotational semantics has been given for the nongraphical synchronous language Esterel. An operational description for this language can be found in [2]. A Statechart-like graphical language is described in [25]. The formal definition of this language uses a process algebra in combination with notions from Esterel.

This paper is structured as follows. In Section 2 we introduce Statecharts informally, illustrated by an example of an unreliable device and a mission control computer in an avionics system. A formal syntax of the language is presented in

Section 3. Section 4 contains a first, straightforward, attempt to axiomatize Statecharts. We show by examples that this axiomatic system does not completely correspond to the intended meaning of Statecharts as described in Section 3. Therefore we slightly modify the specification language, and in Section 5 the final axiomatization is formulated, consisting of an axiom for each basic statechart and a rule for each of the syntactic operators. A denotational semantics for Statechart is defined in Section 6. Section 7 contains details about the property-based specification language and its formal interpretation. Soundness of the axiomatic system with respect to the denotational semantics is proved in Section 8. Moreover, in Section 9 we prove relative completeness of the proof system. Finally, in Section 10, we discuss future extensions.

2. Informal introduction to Statecharts

Realizing the intuitive and pictorial appeal of state-transition diagrams for finite state machines, Statecharts has been designed on the basis of such diagrams. But it is free from the limitations of state machines, such as sequentially, unstructuredness and exponential growth of states when describing concurrency. Indeed, statecharts are exponentially more succinct than state machines, as has been shown in [5]. This is achieved by adding hierarchy, concurrency and broadcast communication. Quoting [12],

Statecharts = state-transition diagrams + depth + orthogonality + broadcast.

The language is built upon so-called *basic states*, like the states in a finite state machine, and transitions. Depth is obtained by allowing *superstates* that contain substates and internal transitions. This leads to a hierarchy of states, corresponding to AND/OR decomposition. Accordingly, there are two types of superstates, AND-states and OR-states. If the system is in an AND-state then it is simultaneously in all its immediate substates. To be in an OR-state means to be in exactly one of its immediate substates. When a superstate is exited, any computation inside is terminated. The immediate substates of an AND-state are called *orthogonal* components and they are executed in parallel. Orthogonal components interact with each other and with their environment by means of events, i.e. signals without measurable duration. Transitions have labels to specify when a transition is enabled and which events are generated when a transition is taken. Events generated by a transition in one orthogonal component are broadcast, possibly triggering transitions in other components which, in turn, might produce new events, etc. Thus a single event can give rise to a whole chain of events.

Execution of a statechart proceeds in (time) steps. In each step a maximal set of enabled transitions is taken with at most one transition per orthogonal component. All events generated by these transitions are assumed to take place simultaneously. This allows us to abstract from the internal chain of transitions and generated events

within a single step. The general idea is that staying in a state takes some time, whereas taking a transition is assumed to be instantaneous. This assumption is essentially Berry's synchrony hypothesis for the language Esterel [2]. This synchrony hypothesis might introduce causal paradoxes, like an event causing itself. In Esterel causal paradoxes are syntactically disallowed whereas in Statecharts causal relationships are respected and paradoxes are removed semantically.

Example 2.1. To introduce and illustrate the language Statecharts, we specify parts of a generalized avionics system. Our example has been inspired by the description of such a system in [29] where it was specified in CSP, VDM and temporal logic. Here we use Statecharts to specify parts of the mission control computer. On request this control function provides aircraft flight data such as best available aircraft position, ground speed, true airspeed and altitude. In addition, information is displayed to the air crew. The requested data is obtained from devices (e.g. radar) or, if a device is broken, data of sufficient accuracy is computed by means of previously stored information. Since several devices have the same interface, we describe the control of a general, abstract, device. First we show, in Fig. 1, how an unreliable device can be modelled in Statecharts. Note the hierarchy of states in this statechart, consisting of OR-states and basic states. For instance, superstate *device* is an OR-state, since the system is either in state *normal* or in *error*. Observe that if the system is in basic state *producing* then it is, simultaneously, in the superstates *normal* and *device*.

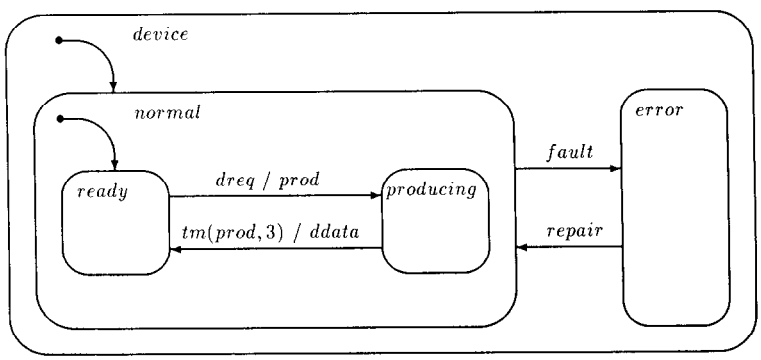

Fig. 1. Unreliable device.

To model the failure of the device and its possible repair, we use external events *fault* and *repair*. The failure hypothesis of the unreliable device can be expressed as an assumption about the expected occurrences of the events *fault* and *repair*. Event *fault* triggers the transition from *normal* to *error*. By taking this transition superstate *normal* is exited explicitly and, simultaneously, its current substate (*ready* or *producing*) is implicitly exited. If event *repair* occurs in basic state *error*, the transition to *normal* is taken and *normal* is entered explicitly. To indicate which substate of *normal* should be entered, there is a *default transition*, drawn as a

transition with no source state, pointing to *ready*. Hence, by taking the transition with label *repair* both *normal* and *ready* are entered. If in state *ready* event *dreq* occurs, expressing an external request for device data, then the transition to *producing* is enabled. When this transition is taken event *prod* is generated and the state *producing* is entered. If, subsequently, in the next two steps event *prod* is not generated then the *time-out* event $tm(prod, 3)$ occurs and event *ddata* is produced.

Transitions have labels of the form "*event-expression/action*". The *event-expression* is a boolean expression involving atomic events. These events can be generated by the outside world, as an input to the statechart, as well as by the statechart itself. The event-expression specifies when the transition is enabled. Let a and b be atomic events. A transition with event-expression a is enabled when event a is generated somewhere in the system (i.e. a statechart and its environment). Event part $a \wedge b$ (resp., $a \vee b$) expresses that the transition can be taken in a step if both a and b (resp., a or b) are generated somewhere in the system in this step. λ is a special event that occurs (by definition) in every step. $tm(e, n)$ denotes a *time-out* event that is generated at a particular step if the last occurrence of event e happened n time steps earlier. In our syntax of event expressions we also allow the *negation* of an event e, denoted by $\neg e$, where e can be λ, an atomic event or a time-out event. The *action* of a label is a set of atomic events that are generated when the transition is taken. Henceforth, a singleton is denoted by its element, and the empty set is omitted. For instance, in our example we use labels *dreq/prod* and *fault* as abbreviations of, respectively, *dreq*/{*prod*} and *fault*/$\emptyset$.

For simplicity we do not consider the general syntax of labels as given in [15]. There a label includes an additional condition part and variable assignments are allowed in actions. Moreover, there are special events to signal entry and exit of a state. Our axiomatic system can be easily adapted to the general case, see Section 10 for an example of these extended labels. Furthermore, in [15] an action is a list of the form $a_1, \ldots, a_n$ whereas we use a set containing these events.

Example 2.2. The avionics system described in [29] contains a control unit which should provide a reliable service based on the unreliable device modelled in Fig. 1. After an external request event *req* this control unit should provide *data* within, say, 30 steps, despite failures of the unreliable device. Therefore it is assumed that, if the device does not respond in time, data of sufficient accuracy can be computed from previous data (such as aircraft position, velocity, etc.). Moreover, the control unit should display the current status of the device and other information to the air crew. The required behaviour is specified in Fig. 2. Observe that *control* is an OR-state, with basic state *off* as its default state. When event *start* occurs the system enters state *on*. As indicated by the two dashed lines, the state *on* consists of three orthogonal components. Entering *on* results in entering all orthogonal components. By the default transitions this means that the states *idle*, *working* and *noflash* are entered and all three components are concurrently active.

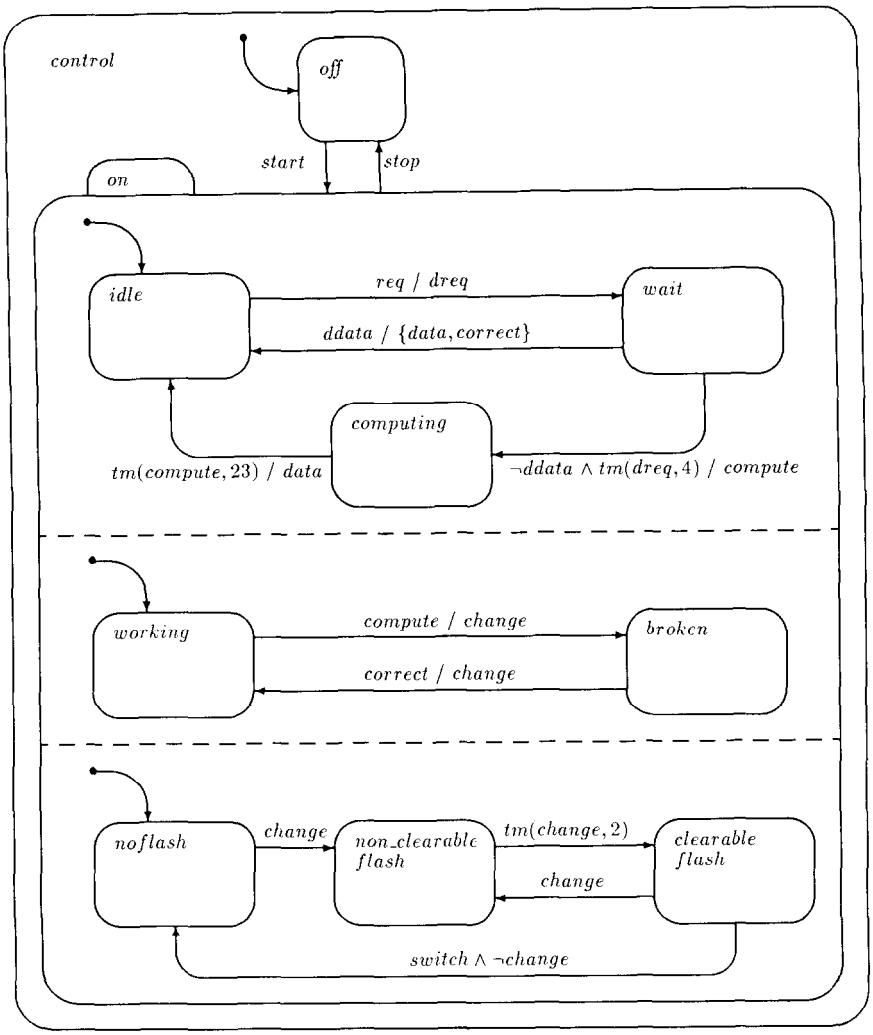

Fig. 2. Mission control computer.

Consider the first component and assume the system is in state *idle*. Event *req* triggers the transition to *wait* and by generating event *dreq* the device is requested for data (see Fig. 1). If event *ddata* is produced within 4 steps then the transition to *idle* is taken, generating *data* and *correct*. Otherwise, event *compute* is generated and *computing* is entered. The second orthogonal component of *on* records the status of the device. This information is displayed to the air crew. If this status changes from *working* to *broken*, or vice versa, internal event *change* is generated. This event is used in the third orthogonal component which specifies how the information is displayed. In case of a change the information is flashed for at least 2 steps. If no new *change* event occurs after these 2 steps then the crew can trigger the transition to *noflash* by means of a switch. Removing $\neg$*change* in the label of

the transition to *noflash* would lead to a nondeterministic choice between the two outgoing transitions of *clearable flash* if the events *change* and *switch* occur simultaneously. By including $\neg$*change* we remove this nondeterminism and we give priority to *change*. Finally, observe that an explicit exit of *on* via the transition with label *stop* leads to an implicit exit of all orthogonal components.

In general, a state can be entered/exited either *explicitly*, by taking a transition connected to the state, or *implicitly*, because an orthogonal component, superstate or substate is entered/exited. Entering an AND-state (resp., OR-state) results in entering all (resp., exactly one) of its immediate substates implicitly. Although not present in this example, it is possible to enter/exit a state inside an orthogonal component explicitly. This leads to an implicit entry/exit of the surrounding AND-state and the other components. Furthermore, more than one orthogonal component can be entered directly via a single, forked, transition. Transitions between orthogonal components are not allowed (e.g. in Fig. 2 no transitions are allowed between *idle* and *working*).

Although the meaning of a statechart is usually intuitively clear, there are a few cases where the specified behaviour is not completely obvious. The first class of problems has to do with causality and the synchrony hypothesis, i.e. the assumption that all transitions within a step are instantaneous and occur simultaneously. This is illustrated by the examples in Fig. 3. By taking a transition with label a/a event a is generated simultaneously. Hence, one could argue that such a transition can always be taken, even if a is not generated externally, since the transition generates its own trigger when it is taken. In our opinion, however, such a transition should not be taken when a is not generated externally. The second example in Fig. 3 indicates that this problem might occur in more complicated situations. It shows a causal loop: a causes b whereas b causes a. In our intended semantics no transition is taken when neither a nor b is generated externally.

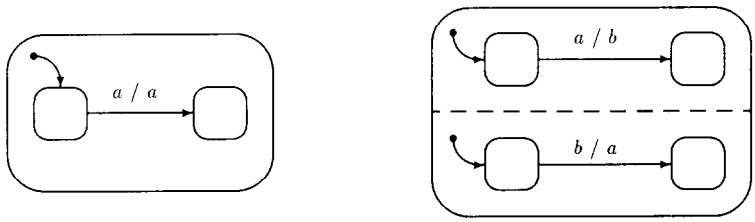

Fig. 3. Causality.

The second class of problems is caused by the possibility to use the negation of events in event-expressions. This might lead to causal paradoxes, as demonstrated in Fig. 4. Consider the transition with label $\neg a/a$. If a is not generated externally, then this transition is enabled. But by taking this transition in a step, event a is generated in the same step, and then the transition would not be enabled in this

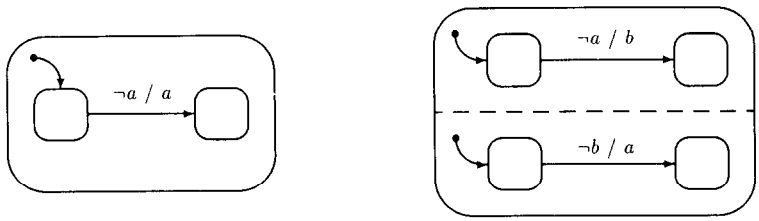

Fig. 4. Negation.

step. The examples from Fig. 4 show that the meaning of negation must be defined carefully. In several papers, e.g. [15, 20], "not" is interpreted as "not yet". In that approach a single computation step is divided into a sequence of micro-steps, corresponding to the sequence of transitions taken in that step. Then a transition with event-expression $\neg a$ is enabled if a has not been generated earlier in the sequence. With this interpretation the meaning of a statechart will, in general, depend on the order in which the transitions inside a step are taken. To obtain a semantics in which the order of the transitions is irrelevant, we follow the suggestions given in [30] where "not" means "never"; a transition with event-expression $\neg a$ is only enabled in a step if a is never generated inside this step. Hence, in our intended semantics a transition with label $\neg a/a$ is never taken.

3. Syntax of statecharts

As a basis for our compositional axiomatization, we formulate in this section a textual syntax for Statecharts. By means of this syntax complete statecharts can be obtained by means of intermediate objects that may have transitions without either source or target states (see, for example, the basic element in Fig. 5). Henceforth, we use the word "statechart" for both these intermediate objects and complete statecharts. First we informally describe the syntax, starting with the primitive objects.

- *Basic statecharts* $[I, O, N]$, where N is a state name, I a set of incoming transitions and O a set of pairs of the form $(T, E/A)$, where T is a transition name and E/A is an event-expression/action pair. Note that only the outgoing transitions are labelled (see Fig. 5).

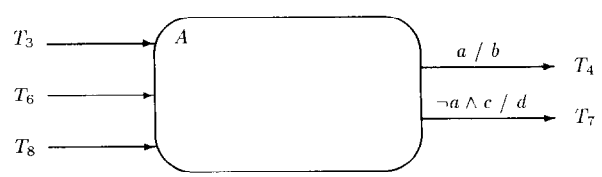

Fig. 5. Basic statechart $[I, O, A]$ with $I = \{T_3, T_6, T_8\}$ and $O = \{(T_4, a/b), (T_7, \neg a \wedge c/d)\}$.

We have the following compound constructs, where B is a basic statechart, S, S_1, S_2 are statecharts, T, T_1, T_2 are transition names, and a is the name of an atomic event.

- *Statification* $Stat(B, S, T)$ makes B a superstate with S inside it and the incoming transition T of S as its default (see Fig. 6, where we use a dashed box to represent an arbitrary statechart). Note that we use the word "statification" different from earlier versions of [11] where it has been used for the act of preparing statecharts. Here it is used for clustering states.
- *Or-construct* $Or(S_1, S_2)$ leads to a statechart that becomes an OR-state after statification.
- *And-construct* $And(S_1, S_2)$ yields an AND-state after statification.

In the constructs above both constituents should not have joint incoming or joint outgoing transitions with the same name, except for the And-construct where joint incoming transitions are allowed.

- *Connect* $Connect(S, T_1, T_2)$ results in a statechart identical to S except that outgoing transition T_1 and incoming transition T_2 of S are connected to form a single complete transition (see Fig. 7).
- *Hide-Closure* $HiCl(S, a)$ hides any generation of a by S (Hiding) and makes S insensitive to any a generated by the environment (Closure).

3.1. Formal syntax of Statecharts

After the informal introduction to the syntax of Statecharts above, we give the formal syntax. First we define the labels of transitions. Let E_e be a set of elementary

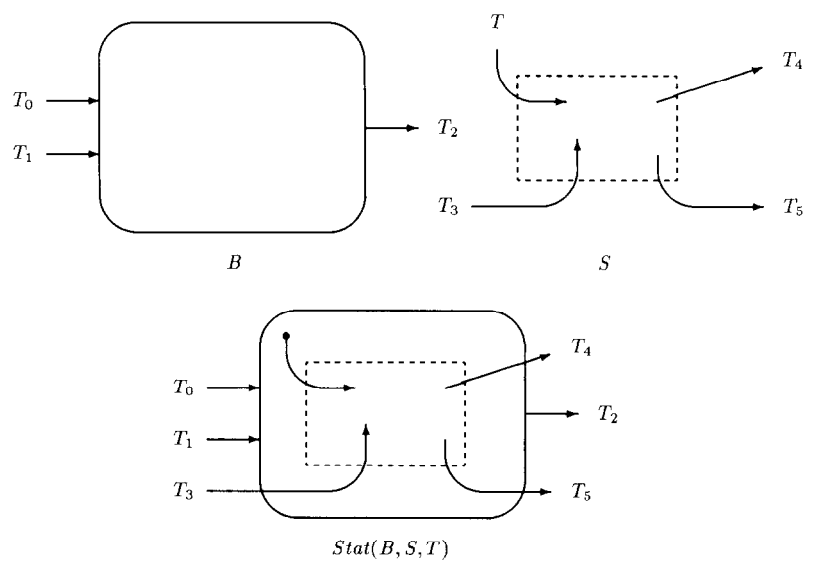

Fig. 6. Statification.

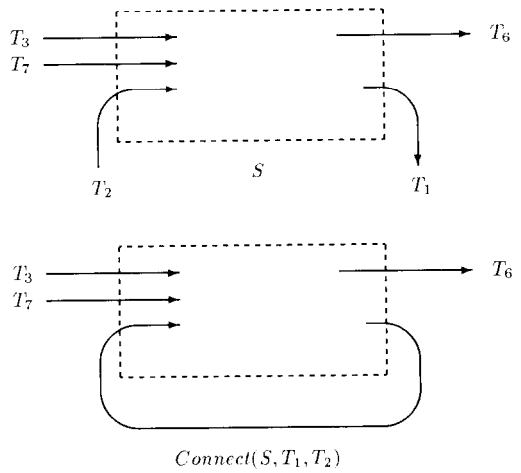

Fig. 7. Connection.

atomic events. Recall that λ is a special event which occurs in every step. $\mathbb{N}$ denotes the set of natural numbers (including 0). The set of event-expressions *Exp* is defined inductively as the least set satisfying

- $\lambda \in Exp$, $\neg\lambda \in Exp$.
- if $a \in E_e$ then $a \in Exp$, $\neg a \in Exp$.
- if $e \in Exp$, $n \in \mathbb{N}$ then $tm(e, n) \in Exp$, $\neg tm(e, n) \in Exp$.
- if $e_1, e_2 \in Exp$ then $e_1 \vee e_2 \in Exp$, $e_1 \wedge e_2 \in Exp$.

The set *Lab* of all symbols that can label the transitions of a statechart is defined by

$$Lab = \{E/A \mid E \in Exp, A \subseteq E_e, A \text{ is finite}\}.$$

Let Σ be the set of state names and $\mathcal{T}$ be the set of transitions. The set of statecharts is defined by the following BNF-grammar, with $a \in E_e$, $\{T, T_1, T_2\} \subseteq \mathcal{T}$, $N \in \Sigma$, $I \subseteq \mathcal{T}$ and $O \subseteq \mathcal{T} \times Lab$, where I and O are finite.

$$S ::= Disj \mid Conj$$

$$Disj ::= Prim \mid Or(Disj, Disj) \mid Connect(Disj, T_1, T_2)$$

$$Conj ::= Default \mid And(Default, Conj)$$

$$Prim ::= Basic \mid Default \mid HiCl(S, a)$$

$$Default ::= Stat(Basic, S, T)$$

$$Basic ::= [I, O, N].$$

Unlike [11], we attach a default transition to every superstate; so also to AND-states. For a set $O \subseteq \mathcal{T} \times Lab$ we use O_T for the set of all transitions in O. For instance,

if $O = \{(T_4, a/b), (T_7, \neg a \wedge c/d)\}$ then $O_T = \{T_4, T_7\}$. Henceforth we use $\equiv$ to denote syntactic equality.

3.2. Syntactic restrictions

In order to describe syntactic restrictions for statecharts, we define two functions IN and OUT. For a given statechart S, $IN(S)$ and $OUT(S)$ are the sets of, respectively, incoming and outgoing transitions of S as defined by Table 1. Then we have the following syntactic restrictions.

- For $[I, O, N]$: $I \cap O_T = \emptyset$ and each transition occurs at most once in O.
- For $Or(S_1, S_2)$: $IN(S_1) \cap IN(S_2) = \emptyset$ and $OUT(S_1) \cap OUT(S_2) = \emptyset$.
- For $Connect(S, T_1, T_2)$: $T_1 \in OUT(S)$ and $T_2 \in IN(S)$.
- For $And(S_1, S_2)$: $OUT(S_1) \cap OUT(S_2) = \emptyset$.
- For $Stat(B, S, T)$: $T \in IN(S)$, $IN(B) \cap IN(S) = \emptyset$ and $OUT(B) \cap OUT(S) = \emptyset$.

Table 1

	IN	OUT
$[I, O, N]$	I	O_I
$Or(S_1, S_2)$	$IN(S_1) \cup IN(S_2)$	$OUT(S_1) \cup OUT(S_2)$
$Connect(S, T_1, T_2)$	$IN(S) - \{T_2\}$	$OUT(S) - \{T_1\}$
$And(S_1, S_2)$	$IN(S_1) \cup IN(S_2)$	$OUT(S_1) \cup OUT(S_2)$
$Stat(B, S, T)$	$(IN(B) \cup IN(S)) - \{T\}$	$OUT(B) \cup OUT(S)$
$HiCl(S, a)$	$IN(S)$	$OUT(S)$

Remarks. (1) In $And(S_1, S_2)$, the intersection of $IN(S_1)$ and $IN(S_2)$ need not be empty. Incoming transitions with identical names are merged into a forked transition.

(2) The *Concat* operation given in [20] has not been provided in our syntax. It can be considered as a derived operation:

$$Concat(S_1, S_2, T_1, T_2) \equiv Connect(Or(S_1, S_2), T_1, T_2).$$

4. Discussion of the axiomatization

In this section we discuss the main problems in defining a compositional axiomatic system for Statecharts that matches with the operational intuition described in Section 2. Therefore we first describe in Section 4.1 a simple assertion language in which properties of statecharts can be expressed. Then, in Section 4.2, we sketch a straightforward axiomatic system. Next we show in Section 4.3, that in some cases this axiomatization does not correspond to our intuitive operational meaning.

Finally, we describe modifications to the assertion language to deal with these problems. The resulting axiomatization is then given in the next section.

4.1. Assertion language

To express properties of a statechart we use a first-order assertion language. The main point in the definition of the assertion language is the choice of the primitives. They should be such that we can express which events occur in the system at a certain step and which events are generated by the statechart under consideration. Furthermore we should be able to express entry and exit of a statechart. Hence, as a first attempt, we use the following primitives to specify properties of a statechart S:

- $occ(a, n)$ to express that event a occurs in step n (generated by S or its environment);
- $G(n)$ to denote the set of events generated by S in step n;
- *st*, ranging over $\mathbb{N}$, denoting the start step, i.e. the step at which S has been entered;
- *es*, ranging over $\mathbb{N} \cup \{\infty\}$, to denote the exit step or ∞ if S is never exited;
- *in*, ranging over $\mathcal{T}_{in} \cup \{\star\}$, the incoming transition, or $\star$ when S is entered implicitly;
- *out*, ranging over $\mathcal{T}_{out} \cup \{\star, \perp\}$, the outgoing transition, or $\star$ when S is exited implicitly, or $\perp$ when S is not exited.

Our assertion language contains two kinds of variables, the above-mentioned *reserved* symbols (*occ*, *G*, *st*, *es*, *in*, *out*), and *logical* variables. We use three types of logical variables:

- Logical $\mathbb{N}$-variables, ranging over the natural numbers.
  Typical symbols are k, l, m, n, ...
- Logical transition variables, ranging over transition names.
  Typical symbols are t, t_1, t_1^i, t_1^o, ...
- Logical G-variables, ranging over functions from $\mathbb{N}$ to sets of events.
  Typical symbols are g, g_1, ...

In addition, the assertion language includes first-order arithmetic. Quantification is only allowed over logical $\mathbb{N}$-variables, and the conventional logical connectives $\neg$, $\vee$, $\wedge$ and $\rightarrow$ are used. We use $\varphi[exp/var]$ to denote the substitution of each free occurrence of variable *var* by expression *exp*. For this assertion language we have the following axioms:

- $st < es$ (The start step is less than the exit step, since a state cannot be entered and exited within a single step.)
- $es = \infty \leftrightarrow out = \perp$ (The exit step equals ∞ if there is no exit.)
- $a \in G(n) \rightarrow occ(a, n)$ (If event a is generated in step n then a occurs in step n.)
- $\forall n: (st \leqslant n \vee n < es) \rightarrow G(n) = \emptyset$. (Nothing is generated before or within the entry step and after the exit step.)

The formal interpretation of assertions is defined in Section 7, using the semantic model that will be given in Section 6. The semantic domain will be such that the axioms above are valid.

As already mentioned in the introduction, we use formulae of the form S **sat** φ to express that statechart S satisfies assertion φ. In Section 7 we formally define when such a correctness formula is valid. Informally, S **sat** φ is valid if φ holds for every possible execution of S.

Example 4.1. For the basic statechart with name *wait* in Fig. 2 we have

$$[\{T_3\}, \{(T_2, ddata/\{data, correct\}),$$
$$(T_5, \neg ddata \wedge tm(dreq, 4)/complete)\}, wait]$$
$$\textbf{sat } in = T_3 \wedge occ(ddata, st+1) \rightarrow out = T_2 \wedge G(st+1) = \{data, correct\}$$
$$\wedge es = st+1.$$

Using our assertion language, we can express real-time safety properties such as

"event a occurs within five steps" $\equiv \exists n < 5$: $occ(a, st+n)$

and

"never exit" $\equiv es = \infty$.

But also liveness properties can be expressed, for instance,

"eventually event a will occur" $\equiv \exists n$: $occ(a, st+n)$,

"eventually exit" $\equiv es \neq \infty$, and

"event a is generated infinitely often" $\equiv \forall k\, \exists n \geq k$: $a \in G(n)$.

4.2. Outline of the axiomatization

The axiomatization consists of an axiom schema for basic statecharts, a proof rule for each of the operators mentioned earlier, and a consequence rule. In this section we give an outline of a straightforward axiomatization using the assertion language described above. In Section 4.3 we show that slight modifications are required for a satisfactory treatment of the examples concerning causality and negation given in Section 2.

First we have an axiom for a basic statechart $[I, O, N]$. Any computation of $[I, O, N]$ enters state N either implicitly (denoted by $\star$) or via a transition in I, and starts waiting to exit N. Then there are three possible situations: either it waits forever, not being able to take any of the outgoing transitions (represented by assertion $WAIT(O, k)$, defined below), or it exists the statechart at a certain step either implicitly (denoted by $\star$), or explicitly by taking an outgoing transition in O_T (represented by $FIRE(O, es)$). This leads to the following axiom:

Axiom 4.2.

$$[I, O, N] \textbf{ sat } (in \in I \vee in = \star) \wedge \forall k, st < k < es\text{: } WAIT(O, k)$$
$$\wedge [(es = \infty) \vee (out = \star \wedge G(es) = \emptyset) \vee FIRE(O, es)].$$

For a transition $t \in O_T$ we use E_t/A_t to denote the label of t in O. Assertion $WAIT(O, k)$ describes that the basic statechart waits at step k. During waiting no event is generated and no transition in O_T can be taken. Hence,

$$WAIT(O, k) \equiv (G(k) = \emptyset) \wedge \bigwedge_{t \in O_T} \neg occ(E_t, k),$$

where $occ(E_t, k)$ is a natural extension of occ for an event-expression E_t, given by

$$occ(\lambda, k) \equiv true,$$

$$occ(\neg a, k) \equiv \neg occ(a, k),$$

$$occ(e_1 \vee e_2, k) \equiv occ(e_1, k) \vee occ(e_2, k),$$

$$occ(e_1 \wedge e_2, k) \equiv occ(e_1, k) \wedge occ(e_2, k),$$

$$occ(tm(e, n), k) \equiv \begin{cases} occ(e, k-n) \wedge \forall m, k-n < m < k: \neg occ(e, m) & \text{if } k \geq n, \\ false & \text{if } k < n. \end{cases}$$

Observe that, due to the "never" interpretation of negation, we have a simple expression for negated events: $\neg e$ occurs in step k iff e does not occur in step k. Note that

$$occ(tm(e, 0), k) \equiv occ(e, k).$$

Furthermore we have defined

$$occ(tm(e, n), n) \equiv occ(e, 0),$$

since the system is allowed to start at time 0. Assertion $FIRE(O, k)$ expresses the condition for taking a transition $t \in O_T$ at step k. Then $occ(E_t, k)$ must hold and all events in A_t are generated by the basic statechart.

$$FIRE(O, k) \equiv \bigvee_{t \in O_T} [(out = t) \wedge occ(E_t, k) \wedge (G(v) = A_t)].$$

Example 4.3. Consider the basic statechart $S \equiv [\{T_1\}, \{(T_2, \neg a/a)\}, A]$. Then S **sat** $out \neq T_2$, because $FIRE(\{(T_2, \neg a/a)\}, k)$ leads to $(out = T_2) \wedge occ(\neg a, k) \wedge (G(k) = \{a\})$. Since $G(k) = \{a\}$ implies $occ(a, k)$, this gives *false*.

The axiomatic system contains a rule for each of the syntactic operators. We give an outline of the rules that will be used in this section to explain the main problems in defining the axiomatization. First consider the And-construct $And(S_1, S_2)$. Assume S_j **sat** φ_j, for $j = 1, 2$. Then at every computation step the G-set of $And(S_1, S_2)$ is the union of the G-sets of S_1 and S_2. Furthermore we have to express how the entry and exit of $And(S_1, S_2)$ is related to entry and exit of its components S_1 and S_2. This will be explained in the next section. Concentrating on the G-set we obtain a rule of the following form, where g_1 and g_2 are fresh logical G-variables.

Rule 4.4 (*And*).

$$\frac{S_1 \textbf{ sat } \varphi_1, \qquad S_2 \textbf{ sat } \varphi_2 \qquad (\varphi_1[g_1/G, \ldots] \wedge \varphi_2[g_2/G \ldots] \wedge (\forall k\colon G(k) = g_1(k) \cup g_2(k)) \wedge \cdots) \rightarrow \varphi}{And(S_1, S_2) \textbf{ sat } \varphi}.$$

The rule for Statification $Stat(B, S, T)$ is similar to the rule for the And-construct, except for the way of entering. In the rule for the hiding and closure construct $HiCl(S, a)$ we require that every occurrence of event a inside S must be generated by S itself. This leads to the condition $\forall n\colon occ(a, n) \rightarrow a \in G(n)$.

4.3. Problems

We show that the axiomatization sketched above does not always correspond to our operational intuition. Consider statechart $S_1 \equiv [\{T_1\}, \{(T_2, a/a)\}, A]$ (see also the first statechart from Fig. 3). If T_2 is taken in step k then $FIRE(\{(T_2, a/a)\}, k)$ leads to $(out = T_2) \wedge occ(a, k) \wedge (G(k) = \{a\})$. Observe that then the requirement in the rule for $HiCl(S_1, a)$ is fulfilled, and hence we can take T_2 after hiding and closure with respect to event a. Thus, in contrast with the intended semantics, T_2 can be taken without any external occurrence of a. Note, however, that the generation of event a by T_2 is not observable, since a has been generated already somewhere in the system. For instance, the environment cannot distinguish between statechart S_1 and $S_2 \equiv [\{T_1\}, \{(T_2, a/\emptyset)\}, A]$. Therefore, in our final axiomatization we do not use $G(n)$, the set of all generated events by a statechart, but a subset $F(n)$. Informally, an event e is an element of this set $F(n)$ for a statechart S if S is responsible for the first generation of e in the system in step n. Then firing transition T_2 above leads to $occ(a, k) \wedge F(k) = \emptyset$, since a must have been generated earlier inside this step.

To make the axiomatic system compositional, the specification of a statechart S must hold for all potential computations of S. Thus we allow any arbitrary environment, and when composing S with another statechart a number of these potential computations can be removed because more information about the environment is available. For the set $F(n)$ this means that we include all possible subsets of the set of generated events, as far as they are consistent with the event-expression (e.g. $F(k) = \{a\}$ is not consistent with event-expression a). Consider, for instance, $[\{T_3\}, \{(T_4, a/b)\}, B]$. Then firing T_4 leads to either $occ(a, k) \wedge F(k) = \emptyset$ or $occ(a, k) \wedge F(k) = \{b\}$.

This example leads to the second problem. Consider the second statechart of Fig. 3 in which there are two transitions with labels a/b and b/a. Taking the transition with label a/b might lead to $occ(a, k) \wedge F(k) = \{b\}$. Similarly, by taking the transition with label b/a we can obtain $occ(b, k) \wedge F(k) = \{a\}$. After applying rules for And and Statification it is possible to take both transitions with $occ(a, k) \wedge occ(b, k) \wedge F(k) = \{a, b\}$. Now observe that the restrictions for hiding and closure with respect to a and b are fulfilled, and hence it is possible to take both transitions without any external a or b. The problem is that we cannot express the causal relationship

between a and b. Therefore we extend our assertion language with a *causality relation* $a <_k b$ to express that the first generation of a in step k precedes the first generation of b in step k. Then $occ(a, k) \wedge F(k) = \{b\}$ leads to $a <_k b$, because $F(k) = \{b\}$ expresses that S_1 claims to generate b for the first time, and hence a must precede b. Similarly, $occ(b, k) \wedge F(k) = \{a\}$ leads to $b <_k a$. Since the relation $<_k$ will be a strict partial order (i.e. irreflexive and transitive), we cannot have both $a <_k b$ and $b <_k a$, and thus these two possibilities cannot be combined. Hence, $F(k) = \{a, b\}$ is impossible and external events are required to trigger these transitions.

5. Compositional axiomatization

In this section we give a compositional axiomatization for Statecharts. We use the assertion language from Section 4.1 with the modifications from Section 4.3. Thus $G(n)$ is replaced by $F(n)$ and, for atomic events a and b, we add $a <_n b$. Furthermore, instead of logical G-variables we use logical F-variables, ranging over functions from $\mathbb{N}$ to sets of events. Typical symbols are $f, f_1, \ldots$. For two logical variables f_1 and f_2 their point-wise union, denoted by $f_1 \cup f_2$, is defined as $(f_1 \cup f_2)(n) = f_1(n) \cup f_2(n)$, for all $n \in \mathbb{N}$. The point-wise subtraction of the function F and a set $\{a\}$, denoted $F \dot{-} \{a\}$, is given by $(F \dot{-} \{a\})(n) = F(n) - \{a\}$, for all $n \in \mathbb{N}$.

Similar to the previous section, we have a number of axioms for the assertion language. In the properties mentioned in Section 3 the function G is replaced by F and we add two axioms about the causality relation $<_n$, expressing that it is a strict partial order.

$$st < es,$$

$$es = \infty \leftrightarrow out = \perp,$$

$$a \in F(n) \rightarrow occ(a, n),$$

$$\forall n: (st \leq n \vee n < es) \rightarrow F(n) = \emptyset,$$

$$\forall n: \neg(a <_n a) \qquad (<_n \text{ is irreflexive}),$$

$$\forall n: (a <_n b \wedge b <_n c) \rightarrow a <_n c \quad (<_n \text{ is transitive}).$$

The axiomatic system contains, for any statechart S, the usual consequence rule by which an assertion can be weakened.

Rule 5.1 (*Consequence*).

$$\frac{S \textbf{ sat } \varphi, \qquad \varphi \rightarrow \varphi'}{S \textbf{ sat } \varphi'}.$$

The axiom for a basic statechart is obtained by a slight modification of Axiom 4.2. Replacing G by F we obtain the following axiom:

Axiom 5.2 (*Basic Statechart*).

$$[I, O, N] \textbf{ sat } (in \in I \vee in = \star) \wedge \forall k, st < k < es\colon WAIT(O, k)$$
$$\wedge [(es = \infty) \vee (out = \star \wedge F(es) = \emptyset) \vee FIRE(O, es)].$$

Let the label of an outgoing transition $t \in O_T$ be given by E_t/A_t. Then

$$WAIT(O, k) \;=\; (F(k) = \emptyset) \wedge \bigwedge_{t \in O_T} \neg occ(E_t, k).$$

Predicate *FIRE* now asserts that the F-set is a subset of the generated events. Furthermore we have to express that certain causal relations exist between newly generated events and the events that triggered the transition. These relations are represented by assertion $soc(E_t, a, k)$, for $a \in A_t$, defined below. Consequently,

$$FIRE(O, k) \;\equiv\; \bigvee_{t \in O_T} \Big[(out = t) \wedge occ(E_t, k) \wedge (F(k) \subseteq A_t)$$
$$\wedge \bigwedge_{a \in A_t} (occ(a, k) \wedge [a \in F(k) \rightarrow soc(E_t, a, k)]) \Big].$$

Assertion $soc(E_t, a, k)$ provides the necessary causal relation between any event $a \in F(k)$ and the events occurring in E_t. For instance, if $E_t = b$ then $b \prec_k a$, whereas $E_t = \neg b$ should not lead to a relation between b and a, since b does not occur at step v. We define *soc* by

$$soc(\lambda, a, k) \;\equiv\; true,$$

$$soc(b, a, k) \;\equiv\; \begin{cases} b \prec_k a & \text{if } b \not\equiv a, \\ false & \text{if } b \equiv a, \end{cases}$$

$$soc(\neg b, a, k) \;\equiv\; \neg occ(b, k),$$

$$soc(tm(e, n), a, k) \;\equiv\; occ(tm(e, n), k),$$

$$soc(e_1 \vee e_2, a, k) \;\equiv\; soc(e_1, a, k) \vee soc(e_2, a, k),$$

$$soc(e_1 \wedge e_2, a, k) \;\equiv\; soc(e_1, a, k) \wedge soc(e_2, a, k).$$

Example 5.3. Firing a transition T_2 with label a/a in step k leads by the formula above to

$$(out = T_2) \wedge occ(a, k) \wedge (F(k) \subseteq \{a\})$$
$$\wedge (occ(a, k) \wedge [a \in F(k) \rightarrow soc(a, a, k)]).$$

Since $soc(a, a, k) \equiv false$, this leads to $(out = T_2) \wedge occ(a, k) \wedge (F(k) \subseteq \{a\}) \wedge a \notin F(k)$. Hence, we obtain $(out = T_2) \wedge occ(a, k) \wedge F(k) = \emptyset$, which indeed expresses that this transition cannot be responsible for the first generation of a in step k.

To formulate a rule for the Or-construct, note that any computation of $Or(S_1, S_2)$ is a computation from either S_1 or S_2. Then the rule for the Or-construct is given by

Rule 5.4 (*Or*).

$$\frac{S_1 \textbf{ sat } \varphi_1, \qquad S_2 \textbf{ sat } \varphi_2}{Or(S_1, S_2) \textbf{ sat } \varphi_1 \vee \varphi_2}.$$

For the Connect-construct, observe that any execution of $Connect(S, T_1, T_2)$

(1) first enters S via a transition different from T_2, then

(2) takes transitions inside S, and then possibly exits S either

- via a transition different from T_1 (and then also exits $Connect(S, T_1, T_2)$), or
- via T_1, re-enters S via T_2, and repeats (2).

To formulate a rule for this construct, we first define the *concatenation* of two assertions φ_1 and φ_2 with respect to T_1 and T_2. Informally, this is an assertion expressing computations that either

- satisfy φ_1 and do not exit via T_1, or
- can be split up in two parts; a computation satisfying φ_1 that exits via T_1 at a certain step m, followed by a computation satisfying φ_2 that enters via T_2 in the same step m.

Formally, with fresh logic variables m, f_1 and f_2, we define

$$\begin{aligned} conc(\varphi_1, \varphi_2, T_1, T_2) \equiv (\varphi_1 \wedge (out \neq T_1)) \vee (\varphi_1[f_1/F, T_1/out, m/es] \\ \wedge \varphi_2[m/st, T_2/in, f_2/F] \\ \wedge (F = f_1 \dot{\cup} f_2)). \end{aligned}$$

Now in the rule we use an assertion $\varphi(n)$, which has a free logical $\mathbb{N}$-variable n. This assertion $\varphi(n)$ represents the behaviour of a sequence of n copies of S that are connected via T_1 and T_2, allowing arbitrary behaviour after a T_1-exit of the last copy. In particular, $\varphi(0)$ should express arbitrary behaviour, and hence should be satisfied by any computation. Thus in the rule below we require that $\varphi(0)$ is (universally) valid. Furthermore, assuming S **sat** $\hat{\varphi}$, we have that

$$conc(\hat{\varphi}, \varphi(n), T_1, T_2) \rightarrow \varphi(n+1)$$

must hold, where $\varphi(n+1) \equiv \varphi[n+1/n]$. Then a computation of $Connect(S, T_1, T_2)$ satisfies $\varphi(n)$ for all n. This leads to the following rule.

Rule 5.5 (*Connect*).

$$\frac{S \textbf{ sat } \hat{\varphi}, \quad \varphi(0), \quad conc(\hat{\varphi}, \varphi(n), T_1, T_2) \rightarrow \varphi(n+1)}{Connect(S, T_1, T_2) \textbf{ sat } (in \neq T_2) \wedge \forall n\colon \varphi(n)}.$$

with n a logical $\mathbb{N}$-variable not occurring free in $\hat{\varphi}$.

To explain the rule for $And(S_1, S_2)$, assume S_j **sat** φ_j is valid, for $j = 1, 2$. Consider the primitive expressions occurring in φ_1 and φ_2. Observe that the start step *st* must have the same value in these two assertions since both components are entered in the same step. Similarly, they should have the same value for *es*. Since *occ* specifies the events that occur in the complete system, the two specifications must agree on these occurrences. Similarly, $<_n$ expresses relations between events in the total system and should be equal in both assertions. For the other primitives (*in*, *F* and *out*), the two components might specify different values. These remaining primitives are related as follows.

- Entry of $And(S_1, S_2)$ is done either
  - implicitly (notation $\star$), by entering both S_1 and S_2 implicitly, or
  - via a joint transition of S_1 and S_2, or
  - via a transition of S_1 that is not a transition of S_2; then S_2 is entered implicitly.

  Similarly for the symmetric case with S_1 and S_2 interchanged.
- At every computation step the F-set of $And(S_1, S_2)$ is the union of the F-sets of S_1 and S_2.
- Concerning the exit of $And(S_1, S_2)$ we have that either
  - both S_1 and S_2 are exited implicitly (denoted by $\star$), or
  - it is exited via a transition of S_1 (S_2 resp.); then S_2 (S_1 resp.) is exited implicitly, or
  - $And(S_1, S_2)$ is never exited, thus neither S_1 nor S_2 are exited (denoted by $\perp$).

By substituting logical variables for these last three primitives we can take the conjunction of both assertions: $\varphi_1[t_1^i/in, f_1/F, t_1^o/out] \wedge \varphi_2[t_2^i/in, f_2/F, t_2^o/out]$, where $t_1^i, f_1, t_1^o, t_2^i, f_2, t_2^o$ are logical variables. The relations between these primitives, as described above, are expressed by

$$and\_in \equiv (in = t_1^i = t_2^i) \vee (in = t_1^i \wedge t_2^i = \star) \vee (in = t_2^i \wedge t_1^i = \star), \quad \text{and}$$

$$and\_out \equiv (out = t_1^o \neq \perp \wedge t_2^o = \star) \vee (out = t_2^o \neq \perp \wedge t_1^o = \star) \vee (out = t_1^o = t_2^o = \perp).$$

This leads to the following rule.

Rule 5.6 (*And*).

$$\frac{S_1 \textbf{ sat } \varphi_1, \quad S_2 \textbf{ sat } \varphi_2 \qquad (\varphi_1[t_1^i/in, f_1/F, t_1^o/out] \wedge \varphi_2[t_2^i/in, f_2/F, t_2^o/out] \wedge and\_in \wedge and\_out \wedge (F = f_1 \cup f_2)) \rightarrow \varphi}{And(S_1, S_2) \textbf{ sat } \varphi}.$$

with $t_1^i, t_2^i, t_1^o, t_2^o$ fresh logical transition variables, and f_1, f_2 fresh logical F-variables.

The rule for the Statification-construct is similar to the rule for the And-construct, except for the way of entering. Entry of $Stat(B, S, T)$ can be done either

(1) by entering B (via a transition or implicitly) and then entering S via default T, or

(2) by entering S via a transition (different from T) and then implicitly entering B.

Hence, in the rule we use

$$stat\_in \equiv (in = t_1^i \wedge t_2^i = T) \vee (in \neq \star \wedge in \neq T \wedge in = t_2^i \wedge t_1^i = \star),$$

$$stat\_out \equiv (out = t_1^o \neq \bot \wedge t_2^o = \star) \vee (out = t_2^o \neq \bot \wedge t_1^o = \star)$$
$$\vee (out = t_1^o = t_2^o = \bot).$$

Rule 5.7 (*Statification*).

$$\frac{B \textbf{ sat } \varphi_1, \qquad S \textbf{ sat } \varphi_2 \qquad (\varphi_1[t_1^i/in, f_1/F, t_1^o/out] \wedge \varphi_2[t_2^i/in, f_2/F, t_2^o/out] \wedge stat\_in \wedge stat\_out \wedge (F = f_1 \cup f_2)) \rightarrow \varphi}{Stat(B, S, T) \textbf{ sat } \varphi}.$$

with t_1^i, t_2^i, t_1^o, t_2^o fresh logical transition variables, and f_1, f_2 fresh logical F-variables.

Next we give a rule for $HiCl(S, a)$. Assume S **sat** φ is valid. To obtain a specification φ' for $HiCl(S, a)$, the possible computations satisfying φ are restricted to those where S is responsible for every occurrence of a (formally, $occ(a, n)$ implies $a \in F(n)$, for every step n). Next a is hidden; it is removed from F (represented in the rule by a substitution in φ') and we require that φ' should not refer to a. This leads to the following rule.

Rule 5.8 (*Hide-Closure*).

$$\frac{S \textbf{ sat } \varphi, \qquad (\varphi \wedge (\forall n\colon occ(a, n) \rightarrow a \in F(n))) \rightarrow \varphi'[F \dot{-} \{a\}/F]}{HiCl(S, a) \textbf{ sat } \varphi'}.$$

provided a does not occur in φ'.

Example 5.9. Consider the avionics system described by the statecharts from Figs. 1 and 2. We would like to prove that after a request the system provides data within a certain number of steps. Thus, for some constant K,

$$occ(req, st + n) \rightarrow \exists k \leq K\colon occ(data, st + n + k).$$

(We assume that free logical variables, such as n here, are universally quantified.) Since in state *off* all *req* events are ignored, we can only prove this property if the system is in state *on*. Therefore we introduce a predicate $instate(on, k_1, k_2)$ which expresses that the system is in state *on* during the steps k_1 through k_2:

$$instate(on, k_1, k_2) \equiv (out \neq \bot) \wedge \exists m < k_1\colon occ(start, m)$$
$$\wedge \forall l, m < l \leq k_2\colon \neg occ(stop, l).$$

This leads to

$$[instate(on, st + n, st + n + K) \wedge occ(req, st + n)]$$
$$\rightarrow \exists k \leq K\colon occ(data, st + n + k).$$

First we consider a guaranteed quick response in case there are no *fault* events. We show that the system satisfies

$$\varphi \equiv [(\forall m: \neg occ(fault, m)) \wedge instate(on, st+n, st+n+3) \wedge occ(req, st+n)] \rightarrow occ(data, st+n+3).$$

Let D be the statechart from Fig. 1, and define

$$\varphi_d \equiv [(\forall m: \neg occ(fault, m)) \wedge (out \neq \perp) \wedge occ(dreq, st+1+n_1)] \rightarrow occ(ddata, st+1+n_1+3).$$

(Note that *device* enters state *ready* in step *st*, and thus is willing to receive requests at step $st+1$.) Then the device satisfies φ_d, provided *prod* is an external event. That is,

$$HiCl(D, prod) \textbf{ sat } \varphi_d.$$

Let C be the statechart of Fig. 2. If we assume that $occ(dreq, st+1+n_1) \rightarrow occ(ddata, st+1+n_1+3)$ holds, then the transition from *wait* to *computing* in C is never taken. Define

$$\varphi_c \equiv [(\forall n_1: occ(dreq, st+1+n_1) \rightarrow occ(ddata, st+1+n_1+3)) \wedge instate(on, st+n, st+n+3) \wedge occ(req, st+n)] \rightarrow occ(data, st+n+3),$$

then C **sat** φ_c.

Now we prove that $\varphi_d[t_1^\circ/out] \wedge \varphi_c[t_2^\circ/out] \wedge and\_out$ implies φ. Assume

$$(\forall m: \neg occ(fault, m)) \wedge instate(on, st+n, st+n+3) \wedge occ(req, st+n).$$

From $instate(on, st+n, st+n+3)$ we obtain $out \neq \perp$, and by *and_out* this leads to $t_1^\circ \neq \perp$. Together with $\forall m: \neg occ(fault, m)$ and $\varphi_d[t_1^\circ/out]$ this leads to

$$\forall n_1: occ(dreq, st+1+n_1) \rightarrow occ(ddata, st+1+n_1+3).$$

Then from $\varphi_c[t_2^\circ/out]$ we obtain $occ(data, st+n+3)$. Hence we can derive, by the And Rule, $And(HiCl(D, prod), C)$ **sat** φ.

Next we consider the general case in which faults may occur and we can only derive a worst case response time, since some of the data might have to be computed. Define

$$\psi \equiv [instate(on, st+n, st+n+3) \wedge occ(req, st+n)] \rightarrow \exists k \leqslant 27: occ(data, st+n+k).$$

Then ψ can be proved for the control unit C, independent of D, provided event *compute* is not generated externally. Thus $HiCl(C, compute)$ **sat** ψ.

6. Denotational semantics of Statecharts

As mentioned in the introduction, the semantic model associates with a statechart a set of maximal computation histories representing all complete executions. It has been shown in [20] that, besides denotations for events generated in each computation step (the observables) and denotations for entry and exit, the following two additional denotations are necessary and sufficient to obtain a compositional semantics: (1) a set of all events assumed to be generated by the total system (i.e. a statechart and its environment) at each step and (2) a causality relation between generated events. More precisely, a computation *history* h of a statechart S is of the form $h = (\hat{s}, i, f, o, s)$ where

- $\hat{s} \in \mathbb{N}$ models the start step.
- $i \in \mathcal{T}_{in} \cup \{\star\}$ is either an incoming transition or $\star$ to model an implicit entry.
- $f: \mathbb{N} \to \{(F, C, \prec) \mid F \subseteq C$ and $\prec$ a strict partial order on $C\}$ is a function which records the state of affairs for every step n by a triple $(F, C, \prec)$, where
  - F is a subset of the events generated by S. An event is an element of F at step n if S is responsible for the first generation of this event in the chain of transitions taken in step n in the system.
  - C is the set of events generated by the total system in step n.
  - $\prec$ denotes the causal relationship between events generated in the total system.
- $o \in \mathcal{T}_{out} \cup \{\star, \bot\}$ is either an outgoing transition, or $\star$ for an implicit exit, or $\bot$ when there is no exit.
- $s \in \mathbb{N} \cup \{\infty\}$ denotes the exit step.

For a function f as above, the three fields of $f(n)$ are denoted by $f^F(n)$, $f^C(n)$ and $f^{\prec}(n)$. Furthermore, h will denote $(\hat{s}, i, f, o, s)$ and similarly for super- and subscripts: h' denotes $(\hat{s}', i', f', o', s')$, h_1 denotes $(\hat{s}_1, i_1, f_1, o_1, s_1)$, etc.

Let $\mathcal{H} = \{h \mid \hat{s} < s, s = \infty \leftrightarrow o = \bot$ and, for all $n \in \mathbb{N}$, $(\hat{s} \leq n \vee n < s) \to f^F(n) = \emptyset\}$. Then our semantic domain is given by $(\mathcal{D}, \sqsubseteq)$, where $\mathcal{D} = \{D \mid D \subseteq \mathcal{H}\}$ and $D_1 \sqsubseteq D_2$ iff $D_1 \subseteq D_2$ for all $D_1, D_2 \in \mathcal{D}$. Clearly, this domain is a complete lattice with bottom element $\emptyset$ and top element $\mathcal{H}$. We give a denotational semantics of Statecharts by defining a semantic function $\mathcal{M}$ that assigns to any statechart S a set of histories, that is, $\mathcal{M}(S) \in \mathcal{D}$.

Definition 6.1 (*Basic*). Consider a basic statechart $[I, O, N]$. For a transition $j \in O_T$ we denote the label of j in O by E_j/A_j. Using predicates *wait* and *fire*, which are defined below, the semantics of $[I, O, N]$ is given by

$$\mathcal{M}([I, O, N]) = \{h \in \mathcal{H} \mid i \in (I \cup \{\star\}) \wedge \forall v, \hat{s} < v < s\colon \mathit{wait}(O, v) \wedge [(s = \infty) \vee (o = \star \wedge f^F(s) = \emptyset) \vee \mathit{fire}(O, s)]\}.$$

Predicate $\mathit{wait}(O, v)$ characterizes the situation in which none of the transition in O_T can be taken. Then none of the triggers of these transitions evaluate to true and no event is generated by the statechart. Consequently, *wait* is defined as

$$\mathit{wait}(O, v) \equiv f^F(v) = \emptyset \wedge \bigwedge_{j \in O_T} \neg \mathit{inC}(E_j, v)$$

where $inC(E_j, v)$ gives the condition in which E_j evaluates to true at step v. It is inductively defined as follows:

$$inC(\lambda, v) \equiv true,$$

$$inC(a, v) \equiv a \in f^C(v) \quad \text{for } a \in E_e,$$

$$inC(\neg a, v) \equiv \neg inC(a, v),$$

$$inC(e_1 \vee e_2, v) \equiv inC(e_1, v) \vee inC(e_2, v),$$

$$inC(e_1 \wedge e_2, v) \equiv inC(e_1, v) \wedge inC(e_2, v),$$

$$inC(tm(e, n), v)$$

$$\equiv \begin{cases} inC(e, v-n) \wedge \forall v',\ v-n < v' < v: \neg inC(e, v') & \text{if } v \geqslant n, \\ false & \text{if } v < n. \end{cases}$$

Predicate *fire* describes the situation in which one of the outgoing transitions can be taken. Using predicate *str*, defined below, *fire* is given by

$$fire(O, v) \equiv \bigvee_{j \in O_T} \Big[o = j \wedge inC(E_j, v) \wedge (f^F(v) \subseteq A_j \subseteq f^C(v)) \wedge \bigwedge_{a \in A_j} (a \in f^F(v) \rightarrow str(E_j, a, v)) \Big].$$

The predicate *str* gives the necessary causal relation between the events constituting the trigger and the events that are generated as a result of taking the transition. It is given by

$$str(\lambda, a, v) \equiv true,$$

$$str(b, a, v) \equiv \begin{cases} (b, a) \in f^{\prec}(v) & \text{if } b \not\equiv a, \\ false & \text{if } b \equiv a, \end{cases}$$

$$str(tm(e, n), a, v) \equiv inC(tm(e, n), v),$$

$$str(\neg b, a, v) \equiv \neg inC(b, v),$$

$$str(e_1 \vee e_2, a, v) \equiv str(e_1, a, v) \vee str(e_2, a, v),$$

$$str(e_1 \wedge e_2, a, v) \equiv str(e_1, a, v) \wedge str(e_2, a, v).$$

Definition 6.2 (*Or*). The semantics of an Or-construct is the union of the semantics of its constituents.

$$\mathcal{M}(Or(S_1, S_2)) = \mathcal{M}(S_1) \cup \mathcal{M}(S_2).$$

Definition 6.3 (*Connect*). Execution of $Connect(S, T_1, T_2)$ consists of first (a) entering S via a transition different from T_2 and then (b) taking transitions as specified by S and possibly exiting S either via a transition different from T_1, or via T_1 and then re-entering S via T_2 and repeating (b). Given two sets of histories D_1, D_2, we define $CONC(D_1, D_2, T_1, T_2)$ as the set of (i) histories of D_1 that do not exit via T_1, and (ii) histories that consist of a history from D_1 with an exit via T_1, followed by a history of D_2 with an entry via T_2. Formally,

$$\begin{aligned}
&CONC(D_1, D_2, T_1, T_2)\\
&\quad = \{h \mid h \in D_1 \wedge o \neq T_1\}\\
&\qquad \cup \{h \mid \exists h_1 \in D_1, h_2 \in D_2 : \hat{s} = \hat{s}_1 \wedge s_1 = \hat{s}_2 \wedge s = s_2 \wedge i = i_1\\
&\qquad\quad \wedge i_2 = T_2 \wedge o_1 = T_1 \wedge o = o_2 \wedge (f^F = f_1^F \dot{\cup} f_2^F)\\
&\qquad\quad \wedge (f^C = f_1^C = f_2^C) \wedge (f^{\prec} = f_1^{\prec} = f_2^{\prec})\}.
\end{aligned}$$

To obtain $\mathcal{M}(Connect(S, T_1, T_2))$ we consider the largest set D satisfying

$$D = CONC(\mathcal{M}(S), D, T_1, T_2),$$

that is, the greatest fixed point $\nu_X.CONC(\mathcal{M}(S), X, T_1, T_2)$. In order to have such a fixed point definition (see, e.g. [4]), $CONC$ is shown to be monotonic in its second argument. From this set we remove the histories that have an entry via T_2. (Note that such a set D will not contain histories that exit via T_1.) This leads to

$$\mathcal{M}(Connect(S, T_1, T_2)) = del_{T_2}(\nu_X.CONC(\mathcal{M}(S), X, T_1, T_2))$$

where $del_{T_2}(D) = \{h \in D \mid i \neq T_2\}$.

Since $CONC$ is anti-continuous, this greatest fixed point can be obtained by an iteration, and the semantics can be given as the intersection of approximations:

$$\mathcal{M}(Connect(S, T_1, T_2)) = del_{T_2}\left(\bigcap_{k \in \mathbb{N}} D_k\right)$$

where $D_0 = \mathcal{H}$, and for $k \in \mathbb{N}$, $D_{k+1} = CONC(\mathcal{M}(S), D_k, T_1, T_2)$.

Definition 6.4 (*And*). The semantics of $And(S_1, S_2)$ is obtained by merging every pair of histories from S_1 and S_2 that agree on the behaviour of the environment. More precisely,

$$\begin{aligned}
&\mathcal{M}(And(S_1, S_2))\\
&\quad = \{h \mid \exists h_1 \in \mathcal{M}(S_1), h_2 \in \mathcal{M}(S_2) : \hat{s} = \hat{s}_1 = \hat{s}_2 \wedge s = s_1 = s_2\\
&\qquad \wedge (f^C = f_1^C = f_2^C) \wedge (f^{\prec} = f_1^{\prec} = f_2^{\prec}) \wedge (f^F = f_1^F \dot{\cup} f_2^F)\\
&\qquad \wedge [(i = i_1 = i_2) \vee (i = i_1 \wedge i_2 = \star) \vee (i = i_2 \wedge i_1 = \star)]\\
&\qquad \wedge [(o = o_1 \neq \perp \wedge o_2 = \star) \vee (o = o_2 \neq \perp \wedge o_1 = \star) \vee o = o_1 = o_2 = \perp]\}.
\end{aligned}$$

Definition 6.5 (*Statification*). The semantics of $Stat(B, S, T)$ is similar to

$$\mathcal{M}(And(B, S)),$$

except for the way in which the statechart is entered; any entry to B leads to S via the default arc T and a direct entry via T is no longer possible. Consequently, the semantics is given as follows:

$$\begin{aligned}
&\mathcal{M}(Stat(B, S, T))\\
&= \{h \mid \exists h_1 \in \mathcal{M}(B),\ h_2 \in \mathcal{M}(S):\ \hat{s} = \hat{s}_1 = \hat{s}_2 \wedge s = s_1 = s_2\\
&\qquad \wedge (f^C = f_1^C = f_2^C) \wedge (f^{\prec} = f_1^{\prec} = f_2^{\prec}) \wedge (f^F = f_1^F \dot{\cup} f_2^F)\\
&\qquad \wedge [(i = i_1 \wedge i_2 = T) \vee (i = i_2 \neq T \wedge i \neq \star \wedge i_1 = \star)]\\
&\qquad \wedge [(o = o_1 \neq \perp \wedge o_2 = \star) \vee (o = o_2 \neq \perp \wedge o_1 = \star) \vee o = o_1 = o_2 = \perp]\}.
\end{aligned}$$

Definition 6.6 (*Hide and Close*). In the semantics of $HiCl(S, a)$ we first require that every occurrence of a is generated by S. Next a is hidden by removing it from the F-set, and by allowing the C- and $\prec$-field to be arbitrary with respect to a (to model occurrences of a that are independent of S). Define the point-wise subtraction of a set-valued function g and a set A, notation $g \dot{-} A$, as $(g \dot{-} A)(v) = g(v) - A$, for all $v \in \mathbb{N}$. For a function $f^{\prec}$ with, for all v, $f^{\prec}(v) \subseteq E_e \times E_e$, we define the restriction of $f^{\prec}$ to events different from a, notation $f^{\prec}|_a$, as $f^{\prec}|_a = f^{\prec} \dot{-} (E_e \times \{a\}) \dot{-} (\{a\} \times E_e)$. Then the semantics is given by

$$\mathcal{M}(HiCl(S, a)) = hide_a(\{h \mid h \in \mathcal{M}(S) \wedge \forall v:\ a \in f^C(v) \rightarrow a \in f^F(v)\})$$

where

$$\begin{aligned}
hide_a(D) = \{h \in \mathcal{H} \mid \exists h_1 \in D:\ &\hat{s} = \hat{s}_1 \wedge i = i_1 \wedge o = o_1 \wedge s = s_1\\
&\wedge (f^F = f_1^F \dot{-} \{a\}) \wedge (f^{\prec}|_a = f_1^{\prec}|_a)\\
&\wedge (f^C \dot{-} \{a\} = f_1^C \dot{-} \{a\})\}.
\end{aligned}$$

7. Interpretation of specifications

In this section we formally define when a statechart S satisfies an assertion φ, that is, we define when a formula S **sat** φ is valid. Therefore we first give the interpretation of assertions. This interpretation requires a computation history, which assigns values to reserved variables, and a *logical variable environment* that assigns values to logical variables. The value of a logical variable v in a logical environment γ is denoted by $\gamma(v)$. We define the *variant* of an environment γ with respect to a variable x and a value v, denoted by $(\gamma : x \mapsto v)$, as

$$(\gamma : x \mapsto v)(y) = \begin{cases} v & \text{if } x \equiv y,\\ \gamma(y) & \text{if } x \not\equiv y. \end{cases}$$

Next we define when an assertion φ holds in a history $h = (\hat{s}, i, f, o, s)$ and a logical variable environment γ, denoted $[\![\varphi]\!]\gamma h$. We give the main points from the definition of this interpretation, leaving the first-order model, relative to which validity is defined, implicit: it is the standard model of arithmetic throughout this paper. For instance, quantification is defined as follows: $[\![\exists n\colon \varphi]\!]\gamma h$ iff there exists a value $v \in \mathbb{N}$ such that $[\![\varphi]\!](\gamma : n \mapsto v)h$.

The primitives from the assertion language that yield a transition name are interpreted as:

$$[\![in]\!]\gamma h = i, \qquad [\![out]\!]\gamma h = o, \qquad [\![t]\!]\gamma h = \gamma(t).$$

Observe that logical variable environment γ gives the value of logical transition variable t. Similarly, we have $[\![f]\!]\gamma h = \gamma(f)$, for a logical F-variable f.
Next we give the interpretation of primitives yielding a value from $\mathbb{N} \cup \{\infty\}$.

$$[\![st]\!]\gamma h = \hat{s}, \qquad [\![es]\!]\gamma h = s, \qquad [\![n]\!]\gamma h = \gamma(n).$$

This can be extended easily to the definition of $[\![exp]\!]\gamma h$, for any expression *exp*.

Now assume *exp* denotes an expression yielding a natural number (for expressions yielding ∞ we take a default value in the definitions below), then the remaining primitives have the following meaning:

$$[\![F(exp)]\!]\gamma h = f^F([\![exp]\!]\gamma h),$$

$[\![occ(a, exp)]\!]\gamma h$ holds iff $a \in f^C([\![exp]\!]\gamma h)$,

$[\![a <_{exp} b]\!]\gamma h$ holds iff $(a, b) \in f^<([\![exp]\!]\gamma h)$.

An assertion φ is *valid*, denoted by $\models \varphi$, iff $[\![\varphi]\!]\gamma h$ holds for all γ and for all $h \in \mathcal{H}$. Since assertions are interpreted in histories of $\mathcal{H}$ only, we have the following valid assertions:

$$\models st < es, \qquad \models es = \infty \leftrightarrow out = \perp,$$

$$\models \forall n\colon n \leqslant st \vee n > es \rightarrow F(n) = \emptyset, \quad \text{and}$$

$$\models \forall n\colon a <_n b \rightarrow occ(a, n) \wedge occ(b, n).$$

Informally, a formula S **sat** φ is valid if φ holds in any history from the semantics of S. Thus S **sat** φ is *valid*, denoted by $\models S$ **sat** φ, iff $[\![\varphi]\!]\gamma h$ for all γ and for all $h \in \mathcal{M}(S)$.

8. Soundness of the axiomatization

In this section we prove that the axiomatization is *sound*, that is, every formula which can be derived is valid. Let $\vdash S$ **sat** φ denote that S **sat** φ is derivable using the axiomatic system from Section 5. Then to prove soundness we have to show that $\vdash S$ **sat** φ implies $\models S$ **sat** φ. This is proved below by showing that the Basic Statechart Axiom yields a valid specification and that all rules preserve validity.

Consequence Rule

Assume $\models S$ **sat** φ and $\models \varphi \to \varphi'$. We prove $\models S$ **sat** φ'. Let γ be arbitrary and consider $h \in \mathcal{M}(S)$. Then, by $\models S$ **sat** φ, $[\![\varphi]\!]\gamma h$, and thus, using $\models \varphi \to \varphi'$, we obtain $[\![\varphi']\!]\gamma h$.

Basic Statechart Axiom

To prove that the Basic Statechart Axiom is valid, we first prove the following lemmas (for all $e \in Exp$, $a \in E_e$, expression exp, environment γ and history h).

Lemma 8.1. $[\![occ(e, exp)]\!]\gamma h = inC(e, [\![exp]\!]\gamma h)$.

Proof. Easy, by induction on the structure of e. For instance, for $e \in E_e$ both sides evaluate to $e \in f^C([\![exp]\!]\gamma h)$. □

Lemma 8.2. $[\![soc(e, a, exp)]\!]\gamma h = str(e, a, [\![exp]\!]\gamma h)$.

Proof. Straightforward, by induction on the structure of e. For instance,

$$[\![soc(\neg b, a, exp)]\!]\gamma h = [\![\neg occ(b, exp)]\!]\gamma h = \neg [\![occ(b, exp)]\!]\gamma h$$

which equals, by Lemma 8.1, $\neg inC(b, [\![exp]\!]\gamma h)$, and thus $str(\neg b, a, [\![exp]\!]\gamma h)$. □

Lemma 8.3. $[\![WAIT(O, exp)]\!]\gamma h = wait(O, [\![exp]\!]\gamma h)$.

Proof. $[\![WAIT(O, exp)]\!]\gamma h = [\![F(exp) = \emptyset \wedge \bigwedge_{t \in O_T} \neg occ(E_t, exp)]\!]\gamma h$. By the interpretation of assertions and Lemma 8.1 this leads to

$$f^F([\![exp]\!]\gamma h) = \emptyset \wedge \bigwedge_{t \in O_T} \neg inC(E_t, [\![exp]\!]\gamma h),$$

and hence $wait(O, [\![exp]\!]\gamma h)$. □

Lemma 8.4. $[\![FIRE(O, exp)]\!]\gamma h = fire(O, [\![exp]\!]\gamma h)$.

Proof. We have

$$[\![FIRE(O, exp)]\!]\gamma h$$

$$= \left[\!\!\left[\bigvee_{t \in O_T} \left[out = t \wedge occ(E_t, exp) \wedge F(exp) \subseteq A_t \right.\right.\right.$$

$$\left.\left.\left. \wedge \bigwedge_{a \in A_t} (occ(a, exp) \wedge [a \in F(exp) \to soc(E_t, a, exp)]) \right] \right]\!\!\right] \gamma h.$$

Using the interpretation of assertions, Lemma 8.1, and Lemma 8.2, this leads to

$$\bigvee_{t \in O_T} \left[o = t \wedge inC(E_t, [\![exp]\!]\gamma h) \wedge f^F([\![exp]\!]\gamma h) \subseteq A_t \right.$$

$$\left. \wedge \bigwedge_{a \in A_t} (a \in f^C([\![exp]\!]\gamma h) \wedge [a \in f^F([\![exp]\!]\gamma h) \to str(E_t, a, [\![exp]\!]\gamma h)]) \right],$$

and thus

$$\bigvee_{j \in O_I} \Big[o = j \wedge inC(E_j, [\![exp]\!]\gamma h) \wedge f^F([\![exp]\!]\gamma h) \subseteq A_j \subseteq f^C([\![exp]\!]\gamma h)$$

$$\wedge \bigwedge_{a \in A_j} (a \in f^F([\![exp]\!]\gamma h) \rightarrow str(E_j, a, [\![exp]\!]\gamma h)) \Big]$$

which equals $fire(O, [\![exp]\!]\gamma h)$. □

Now we are able to show that the Basic Statechart Axiom is valid. Let γ be arbitrary, and consider $h \in \mathcal{M}([I, O, N])$. Then $i \in (I \cup \{\star\})$, and thus $[\![in \in I \vee in = \star]\!]\gamma h$. Furthermore $\forall v, \hat{s} < v < s$: $wait(O, v)$, which leads by Lemma 8.3 to

$$[\![\forall k, st < k < es\text{: } WAIT(O, k)]\!]\gamma h.$$

From the semantics we also obtain $(s = \infty) \vee (o = \star \wedge f^F(s) = \emptyset) \vee fire(O, s)$. Hence, using Lemma 8.4,

$$[\![(es = \infty) \vee (out = \star \wedge F(es) = \emptyset) \vee FIRE(O, es)]\!]\gamma h.$$

Together this leads to

$$[\![(in \in I \vee in = \star) \wedge \forall k, st < k < es\text{: } WAIT(O, k)$$

$$\wedge [(es = \infty) \vee (out = \star \wedge F(es) = \emptyset) \vee FIRE(O, es)]\!]\gamma h.$$

Or Rule

Assume $\models S_1$ **sat** φ_1 and $\models S_2$ **sat** φ_2. We prove $\models Or(S_1, S_2)$ **sat** $\varphi_1 \vee \varphi_2$. Let γ be arbitrary, and consider $h \in \mathcal{M}(Or(S_1, S_2))$. Then $h \in \mathcal{M}(S_1) \cup \mathcal{M}(S_2)$. Hence by our assumption $[\![\varphi_1]\!]\gamma h$ or $[\![\varphi_2]\!]\gamma h$, and thus $[\![\varphi_1 \vee \varphi_2]\!]\gamma h$.

Connect Rule

Assume $\models S$ **sat** $\hat{\varphi}$, $\models \varphi(0)$ and $\models conc(\hat{\varphi}, \varphi(n), T_1, T_2) \rightarrow \varphi[n+1/n]$, i.e.

$$\models ((\hat{\varphi} \wedge out \neq T_1) \vee (\hat{\varphi}[f_1/F, T_1/out, m/es]$$

$$\wedge \varphi(n)[m/st, T_2/in, f_2/F]$$

$$\wedge F = f_1 \dot{\cup} f_2)) \rightarrow \varphi[n+1/n],$$

where m, f_1, f_2 are fresh logical variables, and n is a logical variable not occurring free in $\hat{\varphi}$. We prove $\models Connect(S, T_1, T_2)$ **sat** $in \neq T_2 \wedge \forall n$: $\varphi(n)$. Let γ be arbitrary, and consider $h \in \mathcal{M}(Connect(S, T_1, T_2))$. Then $h \in \bigcap_{k \in \mathbb{N}} D_k$ with $i \neq T_2$ and where $D_0 = \mathcal{H}$, and $D_{k+1} = CONC(\mathcal{M}(S), D_k, T_1, T_2)$, for $k \in \mathbb{N}$. First we prove the following lemma.

Lemma 8.5. If $h \in D_k$ then $[\![\varphi(n)]\!](\gamma: n \mapsto k)h$ for all $k \in \mathbb{N}$.

Proof. By induction on k.

Basic step: Consider $h \in D_0 = \mathcal{H}$. Then, by validity of $\varphi(0)$, $[\![\varphi[0/n]]\!]\gamma h$, and hence $[\![\varphi(n)]\!](\gamma: n \mapsto 0)h$.

Induction step: Let $h \in D_{k+1} = CONC(\mathcal{M}(S), D_k, T_1, T_2)$. Then there are two possibilities.

(1) $h \in \mathcal{M}(S)$ and $o \neq T_1$. From $\models S$ **sat** $\hat{\varphi}$ we obtain $[\![\hat{\varphi}]\!]\gamma h$, and $o \neq T_1$ implies $[\![out \neq T_1]\!]\gamma h$. Since n does not occur in $\hat{\varphi} \wedge out \neq T_1$, this leads to $[\![\hat{\varphi} \wedge out \neq T_1]\!](\gamma: n \mapsto k)h$. Then by $\models conc(\hat{\varphi}, \varphi(n), T_1, T_2) \rightarrow \varphi[n+1/n]$, we obtain $[\![\varphi[n+1/n]]\!](\gamma: n \mapsto k)h$, and thus $[\![\varphi(n)]\!](\gamma: n \mapsto k+1)h$.

(2) There exist h_1 and h_2 such that $h_1 \in \mathcal{M}(S)$, $h_2 \in D_k$,

$$i = i_1 \wedge o = o_2 \wedge \hat{s} = \hat{s}_1 \wedge s_1 = \hat{s}_2 \wedge s = s_2 \wedge (f^C = f_1^C = f_2^C) \wedge (f^{\prec} = f_1^{\prec} = f_2^{\prec}) \tag{1}$$

and $i_2 = T_2 \wedge o_1 = T_1 \wedge (f^F = f_1^F \dot{\cup} f_2^F)$.

Then, using $\models S$ **sat** $\hat{\varphi}$, we obtain $[\![\hat{\varphi}]\!]\gamma h_1$ and, by the induction hypothesis, $[\![\varphi(n)]\!](\gamma: n \mapsto k)h_2$. Define $\gamma' = (\gamma: f_1 \mapsto f_1^F, f_2 \mapsto f_2^F, m \mapsto s_1)$. Since n does not occur free in $\hat{\varphi}$, we have $[\![\hat{\varphi}[f_1/F, m/es]]\!](\gamma': n \mapsto k)h_1$ and, using $s_1 = \hat{s}_2$,

$$[\![\varphi(n)[m/st, f_2/F]]\!](\gamma': n \mapsto k)h_2.$$

Since $i_2 = T_2$ and $o_1 = T_1$, this leads to

$$[\![\hat{\varphi}[f_1/F, T_1/out, m/es]]\!](\gamma': n \mapsto k)(\hat{s}_1, i_1, \langle f^f, f_1^C, f_1^{\prec}\rangle, o, s)$$

and

$$[\![\varphi(n)[m/st, T_2/in, f_2/F]]\!](\gamma': n \mapsto k)(\hat{s}, i, \langle f^f, f_2^C, f_2^{\prec}\rangle, o_2, s_2).$$

Using (1) this can be written as

$$[\![\hat{\varphi}[f_1/F, T_1/out, m/es]]\!](\gamma': n \mapsto k)(\hat{s}, i, \langle f^f, f^C, f^{\prec}\rangle, o, s)$$

and

$$[\![\varphi(n)[m/st, T_2/in, f_2/F]]\!](\gamma': n \mapsto k)(\hat{s}, i, \langle f^f, f^C, f^{\prec}\rangle, o, s).$$

Thus $[\![\hat{\varphi}[f_1/F, T_1/out, m/es] \wedge \varphi(n)[m/st, T_2/in, f_2/F]]\!](\gamma': n \mapsto k)h$. From $f^F = f_1^F \dot{\cup} f_2^F$ we obtain $[\![F = f_1 \dot{\cup} f_2]\!](\gamma': n \mapsto k)h$. Then $\models conc(\hat{\varphi}, \varphi(n), T_1, T_2) \rightarrow \varphi[n+1/n]$ leads to $[\![\varphi[n+1/n]]\!](\gamma': n \mapsto k)h$. Since the logical variables are fresh, we can replace γ' by γ and obtain $[\![\varphi[n+1/n]]\!](\gamma: n \mapsto k)h$, which is equivalent to

$$[\![\varphi(n)]\!](\gamma: n \mapsto k+1)h. \quad \square$$

Since $h \in D_k$ for all $k \in \mathbb{N}$, Lemma 8.5 implies $[\![\varphi(n)]\!](\gamma: n \mapsto k)h$ for all $k \in \mathbb{N}$, and hence $[\![\forall n: \varphi(n)]\!]\gamma h$. Since $i \neq T_2$ this leads to $[\![in \neq T_2 \wedge \forall n: \varphi(n)]\!]\gamma h$.

And Rule

Assume $\models S_1$ **sat** φ_1, $\models S_2$ **sat** φ_2, and

$$\models(\varphi_1[t_1^i/in, f_1/F, t_1^o/out] \wedge \varphi_2[t_2^i/in, f_2/F, t_2^o/out] \wedge and\_in \wedge and\_out \wedge (F = f_1 \dot\cup f_2)) \rightarrow \varphi \tag{2}$$

where t_1^i, t_2^i, t_1^o, t_2^o, f_1 and f_2 are fresh logical variables.

We prove $\models And(S_1, S_2)$ **sat** φ. Let γ be arbitrary, and consider

$$h \in \mathcal{M}(And(S_1, S_2)).$$

Then there exist histories $h_1 \in \mathcal{M}(S_1)$ and $h_2 \in \mathcal{M}(S_2)$ such that

$$\hat{s} = \hat{s}_1 = \hat{s}_2 \wedge s = s_1 = s_2,$$

$$(f^C = f_1^C = f_2^C) \wedge (f^< = f_1^< = f_2^<) \wedge (f^F = f_1^F \dot\cup f_2^F),$$

$$(i = i_1 = i_2) \vee (i = i_1 \wedge i_2 = \star) \vee (i = i_2 \wedge i_1 = \star), \quad \text{and}$$

$$(o = o_1 \neq \perp \wedge o_2 = \star) \vee (o = o_2 \neq \perp \wedge o_1 = \star) \vee (o = o_1 = o_2 = \perp).$$

From $\models S_1$ **sat** φ_1 and $\models S_2$ **sat** φ_2 we obtain $[\![\varphi_1]\!]\gamma h_1$ and $[\![\varphi_2]\!]\gamma h_2$.

Define $\gamma' = (\gamma : t_1^i \mapsto i_1, f_1 \mapsto f_1^F, t_1^o \mapsto o_1, t_2^i \mapsto i_2, f_2 \mapsto f_2^F, t_2^o \mapsto o_2)$. Then, by the interpretation of assertions, we obtain $[\![\varphi_1[t_1^i/in, f_1/F, t_1^o/out]]\!]\gamma' h_1$ and $[\![\varphi_2[t_2^i/in, f_2/F, t_2^o/out]]\!]\gamma' h_2$. Observe that in the interpretation of $\varphi_1[t_1^i/in, f_1/F, t_1^o/out]$ and $\varphi_2[t_2^i/in, f_2/F, t_2^o/out]$ only the $\hat{s}$-, s-, f^C- and $f^<$-components of the histories h_1 and h_2 are used.

Since the histories h_1 and h_2 are equal in these components, we can replace h_1 and h_2 by h and obtain

$$[\![\varphi_1[t_1^i/in, f_1/F, t_1^o/out]]\!]\gamma' h \quad \text{and} \quad [\![\varphi_2[t_2^i/in, f_2/F, t_2^o/out]]\!]\gamma' h. \tag{3}$$

By the definitions of *and_in* and *and_out* we obtain

$$[\![and\_in]\!]\gamma' h = (i = i_1 \wedge i_2 = \star) \vee (i = i_2 \wedge i_1 = \star) \vee (i = i_1 = i_2)$$

and

$$[\![and\_out]\!]\gamma' h = (o = o_1 \neq \perp \wedge o_2 = \star) \vee (o = o_2 \neq \perp \wedge o_1 = \star) \vee o = o_1 = o_2 = \perp.$$

Hence, by the definition of the semantics,

$$[\![and\_in \wedge and\_out]\!]\gamma' h. \tag{4}$$

From $f^F = f_1^F \dot\cup f_2^F$ we obtain

$$[\![F = f_1 \dot\cup f_2]\!]\gamma' h. \tag{5}$$

Now (3), (4) and (5) lead by (2) to $[\![\varphi]\!]\gamma' h$. Since all logical variables are fresh, the interpretation of φ does not depend on the values of these variables in γ'. Hence we can replace γ' by γ and obtain $[\![\varphi]\!]\gamma h$.

Statification Rule

The soundness proof of the rule for the $Stat(B, S, T)$ construct is similar to the proof above for the And Rule. The only difference is the way of entering the construct: if $h_1 \in \mathcal{M}(B)$ and $h_2 \in \mathcal{M}(S)$ then we have $(i = i_1 \wedge i_2 = T) \vee (i = i_2 \neq T \wedge i \neq \star \wedge i_1 = \star)$. Then for $h \in \mathcal{M}(Stat(B, S, T))$ and γ' as above we obtain that

$$[\![stat\_in]\!]\gamma' h = (i = i_1 \wedge i_2 = T) \vee (i \neq \star \wedge i \neq T \wedge i = i_2 \wedge i_1 = \star)$$

is valid.

Hide-Closure Rule

Assume $\models S$ **sat** φ,

$$\models \varphi \wedge (\forall n\colon occ(a, n) \rightarrow a \in F(n)) \rightarrow \varphi'[F \dot{-} \{a\}/F] \tag{6}$$

and a does not occur in φ'. We prove $HiCl(S, a)$ **sat** φ'. Let γ be arbitrary, and consider $h \in \mathcal{M}(HiCl(S, a))$.

Using the definition of $\mathcal{M}$, there exists a $h_1 \in \mathcal{M}(S)$ such that

$$\forall v\colon a \in f_1^C(v) \rightarrow a \in f_1^F(v), \tag{7}$$

$$\hat{s} = \hat{s}_1 \wedge i = i_1 \wedge o = o_1 \wedge s = s_1, \tag{8}$$

$$f^{\prec}|_a = f_1^{\prec}|_a \wedge f^C \dot{-} \{a\} = f_1^C \dot{-} \{a\}, \tag{9}$$

$$f^F = f_1^F \dot{-} \{a\}. \tag{10}$$

Then $\models S$ **sat** φ leads to $[\![\varphi]\!]\gamma h_1$. From (7) we obtain $[\![\forall n\colon occ(a, n) \rightarrow a \in F(n)]\!]\gamma h_1$. Hence, using (6), $[\![\varphi'[F \dot{-} \{a\}/F]]\!]\gamma h_1$. By (8) this leads to

$$[\![\varphi'[F \dot{-} \{a\}/F]]\!]\gamma(\hat{s}, i, f_1, o, s).$$

Let $\tilde{f}_1$ be such that $\tilde{f}_1^F = f_1^F \dot{-} \{a\}$, $\tilde{f}_1^C = f_1^C$, $\tilde{f}_1^{\prec} = f_1^{\prec}$. Then $[\![\varphi']\!]\gamma(\hat{s}, i, \tilde{f}_1, o, s)$, since interpreting $F \dot{-} \{a\}$ in f_1 is equivalent to interpreting F in $\tilde{f}_1$.

In an assertion we can only refer to the f^C component of a history by means of $occ(b, exp)$ for some $b \in E_e$. Since event a does not occur in φ', the only references to $\tilde{f}_1^C$ in φ' are of the form $occ(b, exp)$ with $b \not\equiv a$. Hence we can replace $\tilde{f}_1^C$ in $[\![\varphi']\!]\gamma(\hat{s}, i, \langle \tilde{f}_1^F, \tilde{f}_1^C, \tilde{f}_1^{\prec}\rangle, o, s)$ by $\tilde{f}_1^C \dot{-} \{a\}$ without changing the validity. Similarly, references to $\tilde{f}_1^{\prec}$ in φ' are of the form $b \prec_{exp} c$ with $b \not\equiv a$ and $c \not\equiv a$. Hence we can replace $\tilde{f}_1^{\prec}$ by $\tilde{f}_1^{\prec}|_a$ and obtain $[\![\varphi']\!]\gamma(\hat{s}, i, \langle \tilde{f}_1^F, \tilde{f}_1^C \dot{-} \{a\}, \tilde{f}_1^{\prec}|_a\rangle, o, s)$. Using (9) and (10) we have $f^F = f_1^F \dot{-} \{a\} = \tilde{f}_1^F$, $f^{\prec}|_a = f_1^{\prec}|_a = \tilde{f}_1^{\prec}|_a$ and $f^C \dot{-} \{a\} = f_1^C \dot{-} \{a\} = \tilde{f}_1^C \dot{-} \{a\}$. This leads to $[\![\varphi']\!]\gamma(\hat{s}, i, \langle f^F, f^C \dot{-} \{a\}, f^{\prec}|_a\rangle, o, s)$. Since a does not occur in φ', we can replace $f^C \dot{-} \{a\}$ by f^C and $f^{\prec}|_a$ by $f^{\prec}$. Hence $[\![\varphi']\!]\gamma(\hat{s}, i, f, o, s) = [\![\varphi']\!]\gamma h$.

9. Relative completeness

In this section we discuss completeness of the axiomatization from Section 5. The proof system is *complete* if every valid formula can be derived, i.e. if $\models S$ **sat** φ then

$\vdash S$ **sat** φ. Observe that the application of most of the rules from the axiomatization of Section 4 requires the proof of (implications between) assertions. Hence a complete axiomatic system for Statecharts would also require a complete axiomatization for assertions. This, however, is impossible by Gödel's incompleteness result for first-order arithmetic. Hence the best we can achieve is a *relatively* complete axiomatic system, which is complete relative to the proof of assertions. We show that, indeed, every valid formula S **sat** φ can be derived under the *relative completeness assumption* that every valid assertion is also derivable.

In the proof of relative completeness below we use the following definitions.

Definition 9.1. For an assertion φ the set of histories satisfying φ, denoted by $[\![\varphi]\!]$, is defined as $[\![\varphi]\!] = \{h \in \mathcal{H} \mid \text{for all } \gamma, [\![\varphi]\!]\gamma h\}$.

Observe that $\models S$ **sat** φ is equivalent to $\mathcal{M}(S) \subseteq [\![\varphi]\!]$.

Definition 9.2. An assertion φ is characteristic for a statechart S if $\mathcal{M}(S) = [\![\varphi]\!]$.

Let $Events(\varphi)$ and $Events(S)$ denote the set of events which syntactically occur in, respectively, assertion φ and statechart S. $\mathrm{FV}(\varphi)$ denotes the set of free logical variables occurring in φ.

The completeness proof is based on the following theorems.

Theorem 9.3 (General Expressibility). *For every statechart S there exists an assertion φ such that φ is characteristic for S, $\mathrm{FV}(\varphi) = \emptyset$ and $Events(\varphi) \subseteq Events(S)$.*

Theorem 9.4 (Expressibility Connect). *For a statechart S, define $D_0 = \mathcal{H}$ and $D_{k+1} = CONC(\mathcal{M}(S), D_k, T_1, T_2)$, for $k \in \mathbb{N}$. Then there exists an assertion $\varphi(n)$ such that for all $k \in \mathbb{N}$, $\varphi[k/n]$ is characteristic for D_k and $\mathrm{FV}(\varphi) = \{n\}$.*

The proof of Theorem 9.3 can be found in Section 9.1, and in Section 9.1.3 we prove Theorem 9.4. These theorems are used (in Section 9.2) to prove the following theorem.

Theorem 9.5 (Derivability strongest specification). *For every statechart S, $\vdash S$ **sat** φ where φ is a characteristic assertion for S.*

With this theorem relative completeness follows easily. Assume $\models S$ **sat** φ. From Theorem 9.5 we obtain $\vdash S$ **sat** $\hat{\varphi}$ with $\hat{\varphi}$ a characteristic assertion for S. Then $[\![\hat{\varphi}]\!] = \mathcal{M}(S) \subseteq [\![\varphi]\!]$, and thus $\models \hat{\varphi} \to \varphi$. By the relative completeness assumption this implication is provable, and hence by the Consequence Rule we obtain $\vdash S$ **sat** φ.

9.1. Expressibility

To prove Theorem 9.3, we have to show that for every statechart S there exists an assertion φ such that $\mathcal{M}(S) = [\![\varphi]\!]$, $\mathrm{FV}(\varphi) = \emptyset$, and $\mathit{Events}(\varphi) \subseteq \mathit{Events}(S)$. In principle we follow the standard way of proving expressibility: first code the denotations from the semantics into natural numbers and show that the semantics of every language construct is recursively enumerable. Next use the fact that recursively enumerable sets are arithmetical, i.e. expressible by a formula in first-order arithmetic [33]. (Thus our notion of completeness is in fact *arithmetical completeness* in the sense of [10].) Often it is not possible to code the denotations directly (e.g. owing to an infinite number of variables), but then it can be shown that the semantics is determined by a finite part of these denotations (e.g. the finite set of variables in the program). In our framework we have a similar problem, due to the f-component of histories:

(a) this function f has an infinite domain (viz. $\mathbb{N}$), and

(b) the C- and $<$-components of $f(n)$ can be infinite.

To cope with (a), we consider approximations of histories. Therefore, for $k \in \mathbb{N}$, we define the *kth approximation* of a history h, denoted by $h{\downarrow}k$. Informally, $h{\downarrow}k$ is the set of all histories that coincide with h for the first k steps.

Definition 9.6. The kth approximation of a history h is defined as

$$\begin{aligned} h{\downarrow}k = \{h_1 \in \mathcal{H} \mid & (\hat{s} > k \to \hat{s}_1 > k) \\ & \wedge (\hat{s} \leq k \to i_1 = i \wedge \hat{s}_1 = \hat{s} \wedge \forall n \leq k : f_1(n) = f(n)) \\ & \wedge (s > k \to s_1 > k) \wedge (s \leq k \to s_1 = s \wedge o_1 = o)\}. \end{aligned}$$

For a set of histories $D \subseteq \mathcal{H}$ we define $D{\downarrow}k = \bigcup_{h \in D} h{\downarrow}k$.

The definition of $h{\downarrow}k$ can be explained by considering three cases:

(1) $\hat{s} > k$, which implies $s > k$. Then h_1 is allowed to be arbitrary as long as it also has a start step and an exit step greater than k.

(2) $\hat{s} \leq k < s$. Then h_1 must have the same entry step as h and the values of f for steps up to and including k must coincide. h_1 is allowed to have a different way of exiting and arbitrary values for f after step k.

(3) $k \geq s$. Then h_1 must be equal to h except for the values of f at steps after k.

In Section 9.1.1 we prove the following lemma.

Lemma 9.7. *For any statechart S, the sequence $\mathcal{M}(S){\downarrow}k$, $k = 0, 1, \ldots$, is a nonincreasing sequence (in the subset ordering) of sets that converges to $\mathcal{M}(S)$, i.e.*

$$\mathcal{M}(S) = \bigcap_{k \in \mathbb{N}} \mathcal{M}(S){\downarrow}k.$$

Note that convergence is achieved thanks to the absence of any fairness constraints in Statecharts. (Fairness constraints require higher ordinals transcending ω [7].)

Although histories in $\mathcal{M}(S)\downarrow k$ have an infinite f-component, they are arbitrary after k. Hence $\mathcal{M}(S)\downarrow k$ is characterized by modified histories which have an f-component that is restricted to $\{0, \ldots, k\}$.

To deal with problem (b), observe that the C-components are arbitrary outside $Events(S)$ which is a finite set. The same observation holds for the $\prec$-components. Let EU denote the set $Events(S)$. Then $\mathcal{M}(S)\downarrow k$ is determined by histories in which the C- and $\prec$-components are restricted to EU. Note that, by the definition of the semantics, the F-component is already a subset of EU. This leads to the following definition.

Definition 9.8. For a history $h = (\hat{s}, i, f, o, s)$ and $k \in \mathbb{N}$, the restricted history $h\downarrow(k, EU)$ is defined by $h\downarrow(k, EU) = (\hat{s}, i, f_1, o_1, s_1)$ where the domain of f_1 is $\{0, \ldots, k\}$, for all v: $0 \leq v \leq k$, $f_1^F(v) = f^F(v)$, $f_1^C(v) = f^C(v) \cap EU$, $f_1^{\prec}(v) = f^{\prec}(v) \cap (EU \times EU)$, if $s \leq k$ then $s_1 = s$, $o_1 = o$ and $s_1 = k+1$, $o_1 = \perp$ otherwise.

Next we extend this definition to sets of histories.

Definition 9.9. For a set D, $D \subseteq \mathcal{H}$, define

$$D\downarrow(k, EU) = \{h\downarrow(k, EU) \mid h \in D \wedge \hat{s} \leq k\}.$$

Above we have intuitively argued that $\mathcal{M}(S)\downarrow k$ can be characterized by restricted histories. This is formalized in the following lemma, which is proved in Section 9.1.2.

Lemma 9.10.

$$\mathcal{M}(S)\downarrow k = \{h \in \mathcal{H} \mid h\downarrow(k, EU) \in \mathcal{M}(S)\downarrow(k, EU)\} \cup \{h \in \mathcal{H} \mid \hat{s} > k\}.$$

Observe that $\{h \in \mathcal{H} \mid \hat{s} > k\}$ can be characterized by assertion $st > k$. Hence, to characterize $\mathcal{M}(S)\downarrow k$ by an assertion, we concentrate on $\mathcal{M}(S)\downarrow(k, EU)$. Since $h\downarrow(k, EU)$ is finite, it can be coded into the natural numbers (by Gödel numbering). Let its code be denoted by $code(h\downarrow(k, EU))$.

Definition 9.11. The coding of $\mathcal{M}(S)\downarrow(k, EU)$ is defined by

$$\mathcal{M}^c(S)\downarrow(k, EU) = \{code(h\downarrow(k, EU)) \mid h\downarrow(k, EU) \in \mathcal{M}(S)\downarrow(k, EU)\}.$$

Then we prove the following lemma.

Lemma 9.12. *For every statechart S, $\{(l, k) \mid l \in \mathcal{M}^c(S)\downarrow(k, EU), k \in \mathbb{N}, l \in \mathbb{N}\}$ is a recursively enumerable set.*

Proof. Since

$$\{(l, k) \mid l \in \mathcal{M}^c(S)\downarrow(k, EU), l \in \mathbb{N}, k \in \mathbb{N}\}$$
$$= \bigcup_{k \in \mathbb{N}} \{(l, k) \mid l \in \mathcal{M}^c(S)\downarrow(k, EU), l \in \mathbb{N}\},$$

it is sufficient to show that $\mathcal{M}^c(S)\downarrow(k, EU)$ is r.e. for all $k \in \mathbb{N}$. Since coding and decoding of $h\downarrow(k, EU)$ is recursive, it is sufficient to prove that $\mathcal{M}(S)\downarrow(k, EU)$ is r.e. for all $k \in \mathbb{N}$. Consider $h \in \mathcal{M}(S)\downarrow(k, EU)$. Since S has a finite number of incoming and outgoing transitions, the fields i and o of h are element of a finite set. Also for $\hat{s}$ we have a finite number of possibilities, since $\hat{s} \in \mathbb{N}$ and $\hat{s} \leq k$. Similarly, since $s \leq k$ or $s = k+1$, the exit step s is an element of a finite set. Finally observe that the function f of h has a finite domain ($\{0, \ldots, k\}$) and for all v, $0 \leq v \leq k$, the components $f_1^F(v)$, $f_1^C(v)$ and $f_1^{\prec}(v)$ are finite (using that $f_1^F(v) \subseteq Events(S)$ and $Events(S)$ is finite). Hence, for all components of h we have a finite number of possibilities, thus $\mathcal{M}(S)\downarrow(k, EU)$ is finite, and consequently $\mathcal{M}(S)\downarrow(k, EU)$ is r.e. for all $k \in \mathbb{N}$. □

Since our assertion language includes first-order arithmetic, recursion theory (see, e.g. [33]) this leads to the following lemma.

Lemma 9.13. *For every statechart S there exists an assertion $\psi(n, m)$ with* $FV(\psi) = \{n, m\}$ *and* $Events(\psi) = \emptyset$ *such that for all* $k, l \in \mathbb{N}$,

$$\models \psi[k/n, l/m] \text{ iff } l \in \mathcal{M}^c(S)\downarrow(k, EU).$$

Note that this lemma implies that, for all h and γ, $[\![\psi[k/n, l/m]]\!]\gamma h$ iff $l \in \mathcal{M}^c(S)\downarrow(k, EU)$. Since our assertion language allows references to all components of histories and includes first-order arithmetic, we can give an assertion which describes the coding of histories, as follows.

Lemma 9.14. *For every statechart S there exists an assertion $\chi(n, m)$ with* $FV(\chi) = \{n, m\}$ *and* $Events(\chi) \subseteq Events(S)$ *such that for all h, γ and $k, l \in \mathbb{N}$,*

$$[\![\chi[k/n, l/m]]\!]\gamma h \text{ iff } code(h\downarrow(k, EU)) = l.$$

Finally we prove Theorem 9.3. Consider a statechart S. From Lemma 9.13 and Lemma 9.14 we obtain assertions ψ and χ. Define

$$\varphi(n) \equiv (\exists m : \psi(n, m) \wedge \chi(n, m)) \vee (st > n),$$

then $FV(\varphi) = \{n\}$ and $Events(\varphi) \subseteq Events(S)$. Now we prove that, for $k \in \mathbb{N}$, $\varphi[k/n]$ characterizes $\mathcal{M}(S)\downarrow k$, that is, $[\![\varphi[k/n]]\!] = \mathcal{M}(S)\downarrow k$. Let γ and $h \in \mathcal{H}$ be arbitrary. Then

$[\![\varphi[k/n]]\!]\gamma h$
iff, by definition of φ,
there exists $l \in \mathbb{N}$ such that $[\![\psi[k/n, l/m]]\!]\gamma h$ and $[\![\chi[k/n, l/m]]\!]\gamma h$, or $\hat{s} > k$
iff, using Lemmas 9.13 and 9.14,
there exists $l \in \mathbb{N}$ such that $l \in \mathcal{M}^c(S)\downarrow(k, EU)$ and $code(h\downarrow(k, EU)) = l$, or $\hat{s} > k$
iff
$code(h\downarrow(k, EU)) \in \mathcal{M}^c(S)\downarrow(k, EU)$ or $\hat{s} > k$
iff, by Definition 9.11 of $\mathcal{M}^c$,
$h\downarrow(k, EU) \in \mathcal{M}(S)\downarrow(k, EU)$ or $\hat{s} > k$

iff, using Lemma 9.10 (recall that $h \in \mathcal{H}$),
$h \in \mathcal{M}(S)\downarrow k$.

Hence, using Lemma 9.7, $[\![\forall n: \varphi(n)]\!] = \bigcap_{k \in \mathbb{N}} \mathcal{M}(S)\downarrow k = \mathcal{M}(S)$.

Furthermore, $FV(\forall n: \varphi(n)) = \emptyset$ and $Events(\forall n: \varphi(n)) \subseteq Events(S)$. This proves Theorem 9.3, since $\forall n: \varphi(n)$ satisfies the required conditions.

9.1.1. Proof of Lemma 9.7

First we prove the following lemma.

Lemma 9.15. *For any statechart S and history $h \in \mathcal{H}$, if there exists a sequence $h_0, h_1, \ldots$ such that $h_k \in \mathcal{M}(S)$ and $h \in h_k\downarrow k$ for all $k \in \mathbb{N}$, then $h \in \mathcal{M}(S)$.*

Proof. Consider $h_0, h_1, \ldots$ with $h_k \in \mathcal{M}(S)$ and $h \in h_k\downarrow k$, for all $k \in \mathbb{N}$.

If $s < \infty$, then choose $l \in \mathbb{N}$ such that $l \geqslant s$. From $h \in h_l\downarrow l$ we obtain $s_l \leqslant l$ (since $s_l > l$ would imply $s > l$), and hence $s = s_l$ and $o = o_l$. Since $\hat{s}_l < s_l \leqslant l$, we obtain $i = i_l$, $\hat{s} = \hat{s}_l$ and $\forall v \leqslant l: f(v) = f_l(v)$. Thus $\forall v \leqslant s: f(v) = f_l(v)$. Note that after s (which is equal to s_l) f^F and f_l^F are both empty. Since $h_l \in \mathcal{M}(S)$ and the semantics contains arbitrary C- and $<$-components after the exit step, also $h \in \mathcal{M}(S)$.

If $s = \infty$ then we prove a stronger property. Let $P(S)$ be the following property:

> if for $h \in \mathcal{H}$ there exists a sequence $h_{j_0}, h_{j_1}, \ldots$
> such that $j_0 < j_1 < \cdots$, $s = \infty$, and for all $k \in \mathbb{N}$: $h_{j_k} \in \mathcal{M}(S)$,
> and $h \in h_{j_k}\downarrow j_k$
> then $h \in \mathcal{M}(S)$.

Note that, for all $k \in \mathbb{N}$, $s = \infty$ implies $s > j_k$, and thus from $h \in h_{j_k}\downarrow j_k$ we obtain $s_{j_k} > j_k$. Furthermore, since $h \in \mathcal{H}$, $s = \infty$ implies $o = \perp$. Then we prove $P(S)$ by induction on the structure of S.

Basic Statechart. $S \equiv [I, O, N]$. Consider $m \in \mathbb{N}$ such that $\hat{s} < m$. Since $j_0 < j_1 < \cdots$ is an increasing infinite sequence, we can find j_k such that $m < j_k$. From $h \in h_{j_k}\downarrow j_k$ and $\hat{s} < j_k$ we obtain $\hat{s}_{j_k} < j_k$, $\hat{s} = \hat{s}_{j_k}$ and $i = i_{j_k}$. By the definition of the semantics, $h_{j_k} \in \mathcal{M}([I, O, N])$ implies that $wait(O, v)$ holds for all v with $\hat{s}_{j_k} < v < s_{j_k}$. From $s_{j_k} > j_k > m$ and $\hat{s}_{j_k} = \hat{s}$, $wait(O, v)$ holds for all v with $\hat{s} < v < m$. By $i = i_{j_k}$ we obtain $i \in (I \cup \{\star\})$. Hence, for all $m \in \mathbb{N}$ with $\hat{s} < m$ we have $\forall v, \hat{s} < v < m$: $wait(O, v)$. Thus $\forall v, \hat{s} < v < \infty$: $wait(O, v)$. Together with $i \in (I \cup \{\star\})$, $s = \infty$ and $o = \perp$, this leads to $h \in \mathcal{M}([I, O, N])$.

Or. $S \equiv Or(S_1, S_2)$. Assume $P(S_1)$ and $P(S_2)$. Since $\mathcal{M}(S) = \mathcal{M}(S_1) \cup \mathcal{M}(S_2)$, there exists an infinite sequence $h_{m_0}, h_{m_1}, \ldots$, which is a subsequence of $h_{j_0}, h_{j_1}, \ldots$, and an $i \in \{1, 2\}$ such that $h_{m_k} \in \mathcal{M}(S_i)$, for all $k \in \mathbb{N}$. By the induction hypothesis this leads to $h \in \mathcal{M}(S_i)$, and hence $h \in \mathcal{M}(Or(S_1, S_2))$.

And. $S \equiv And(S_1, S_2)$. Assume $P(S_1)$ and $P(S_2)$. Since $h_{j_k} \in \mathcal{M}(S)$, we can construct by means of the sequence $h_{j_0}, h_{j_1}, \ldots$ two new sequences $h^1_{j_0}, h^1_{j_1}, \ldots \in \mathcal{M}(S_1)$, and $h^2_{j_0}, h^2_{j_1}, \ldots \in \mathcal{M}(S_2)$, such that $h^1_{j_k}$ and $h^2_{j_k}$ can be combined, by the semantic definition of the And-construct, into h_{j_k}. Similar to the construction of these sequences, we

can construct h^1 and h^2 which by the definition of the semantics of *And* can be combined into h, and for which we have $h^1 \in h^1_{j_k}\downarrow j_k$ and $h^2 \in h^2_{j_k}\downarrow j_k$, for all $k \in \mathbb{N}$. By the induction hypothesis, $h^1 \in \mathcal{M}(S_1)$, and $h^2 \in \mathcal{M}(S_2)$. Thus $h \in \mathcal{M}(And(S_1, S_2))$.

Statification and Hide Closure. The proof for these constructs is similar to the proof for the And-construct.

Connect. $S \equiv Connect(S_1, T_1, T_2)$. Assume $P(S_1)$. Consider a sequence $h_{j_0}, h_{j_1}, \ldots \in \mathcal{M}(S) = del_{T_2}(\bigcap_{l \in \mathbb{N}} D_l)$. Since $h_{j_k} \in D_l$ for all $l \in \mathbb{N}$, it suffices to prove that every D_l has the required property.

- D_0 is trivial, since $h \in \mathcal{H}$.
- Consider $h_{j_0}, h_{j_1}, \ldots \subset D_{l+1} = CONC(\mathcal{M}(S_1), D_l, T_1, T_2)$. Then there exists an infinite subsequence $h_{m_0}, h_{m_1}, \ldots$ such that
  (1) either $h_{m_k} \in \{h \mid h \in \mathcal{M}(S_1) \wedge o \neq T_1\}$ for all k,
  (2) or

$$h_{m_k} \in \{h \mid \exists h_1 \in \mathcal{M}(S_1), h_2 \in D_l: \hat{s} = \hat{s}_1 \wedge s_1 = \hat{s}_2 \wedge s = s_2 \wedge i = i_1 \wedge i_2 = T_2 \wedge o_1 = T_1 \wedge o = o_2 \wedge f^F = f_1^F \cup f_2^F \wedge f^C = f_1^C = f_2^C \wedge f^{\prec} = f_1^{\prec} = f_2^{\prec}\},$$

  for all k.

In the first case we obtain $h \in D_{l+1}$ from the induction hypothesis for S_1. In the second case we can prove $h \in D_{l+1}$ with the same techniques as have been used for the And-construct.

This proves Lemma 9.15. □

Finally we can prove Lemma 9.7, i.e. $\mathcal{M}(S) = \bigcap_{k \in \mathbb{N}} \mathcal{M}(S)\downarrow k$, as follows

(1) Assume $h \in \mathcal{M}(S)$. Since $h \in h\downarrow k$, we obtain $h \in \mathcal{M}(S)\downarrow k$ for all k. Hence $h \in \bigcap_{k \in \mathbb{N}} \mathcal{M}(S)\downarrow k$. Thus $\mathcal{M}(S) \subseteq \bigcap_{k \in \mathbb{N}} \mathcal{M}(S)\downarrow k$.

(2) Assume $h \in \mathcal{M}(S)\downarrow k$ for all $k \in \mathbb{N}$. Then $h \in \mathcal{H}$ and there exists a sequence $h_0, h_1, \ldots$ such that $h_k \in \mathcal{M}(S)$ and $h \in h_k\downarrow k$ for all $k \in \mathbb{N}$. By Lemma 9.15, $h \in \mathcal{M}(S)$. Thus $\bigcap_{k \in \mathbb{N}} \mathcal{M}(S)\downarrow k \subseteq \mathcal{M}(S)$.

9.1.2. *Proof of Lemma 9.10*

We have to prove $\mathcal{M}(S)\downarrow k = \{h \in \mathcal{H} \mid h\downarrow(k, EU) \in \mathcal{M}(S)\downarrow(k, EU)\} \cup \{h \in \mathcal{H} \mid \hat{s} > k\}$. Recall that, by definition, $\mathcal{M}(S)\downarrow k = \bigcup_{h \in \mathcal{M}(S)} h\downarrow k$.

(1) Assume $h \in \mathcal{M}(S)\downarrow k$. Since $\hat{h}\downarrow k \subseteq \mathcal{H}$ for all $\hat{h}$, we have $h \in \mathcal{H}$. If $\hat{s} > k$ then $h \in \{h \in \mathcal{H} \mid \hat{s} > k\}$. If $\hat{s} \leq k$ then, by Definition 9.9, $h\downarrow(k, EU) \in \mathcal{M}(S)\downarrow(k, EU)$. Hence $h \in \{h \in \mathcal{H} \mid h\downarrow(k, EU) \in \mathcal{M}(S)\downarrow(k, EU)\}$.

(2) Assume $h \in \{h \in \mathcal{H} \mid h\downarrow(k, EU) \in \mathcal{M}(S)\downarrow(k, EU)\} \cup \{h \in \mathcal{H} \mid \hat{s} > k\}$. Then there are two possibilities:

- $h \in \mathcal{H}$ and $\hat{s} > k$. Observe that, for every S and k, $\mathcal{M}(S)$ contains a history h_1 with $\hat{s}_1 > k$. From $h \in \mathcal{H}$ and $h_1 \in \mathcal{H}$ we obtain $s \geq \hat{s} > k$ and $s_1 \geq \hat{s}_1 > k$. Hence $h \in h_1\downarrow k$ and thus $h \in \mathcal{M}(S)\downarrow k$.
- $h \in \mathcal{H}$ and $h\downarrow(k, EU) \in \mathcal{M}(S)\downarrow(k, EU)$. Then there exists a $h_1 \in \mathcal{M}(S)$ such that $\hat{s}_1 \leq k$ and $h\downarrow(k, EU) = h_1\downarrow(k, EU)$. By Definition 9.8, this implies $\hat{s} = \hat{s}_1$, $i = i_1$,

for all v, $0 \leq v \leq k$, $f_1^F(v) = f^F(v)$, $f_1^C(v) \cap EU = f^C(v) \cap EU$, $f_1^{\prec}(v) \cap (EU \times EU) = f^{\prec}(v) \cap (EU \times EU)$ and $(s > k \wedge s_1 > k) \vee (s = s_1 \wedge o = o_1)$. Define $A_v = f^C(v) - EU$ and $R_v = f^{\prec}(v) - (EU \times EU)$. Recall that the semantics of S is arbitrary outside EU, i.e. contains every, a priori, possible computation. Hence, since $h_1 \in \mathcal{M}(S)$, there exists a history $h_2 \in \mathcal{M}(S)$ such that $\hat{s}_2 = \hat{s}_1$, $i_2 = i_1$, $s_2 = s_1$, $o_2 = o_1$, and for all v, $f_2^F(v) = f_1^F(v)$, $f_2^C(v) = (f_1^C(v) \cap EU) \cup A_v$, $f_2^{\prec}(v) = (f_1^{\prec}(v) \cap (EU \times EU)) \cup R_v$. Next we prove $h \in h_2 \downarrow k$ by considering the following cases:

- $\hat{s}_2 > k$. Since $\hat{s}_2 = \hat{s}_1$ this implies $\hat{s}_1 > k$, which leads to a contradiction with $\hat{s}_1 \leq k$ above.
- $\hat{s}_2 \leq k$. Then $\hat{s}_1 \leq k$, and thus $\hat{s} = \hat{s}_1 = \hat{s}_2$, $i = i_1 = i_2$, and for all v, $0 \leq v \leq k$,

$$f^F(v) = f_1^F(v) = f_2^F(v),$$

$$f^C(v) = (f^C(v) \cap EU) \cup A_v = (f_1^C(v) \cap EU) \cup A_v = f_2^C(v),$$

$$f^{\prec}(v) = (f^{\prec}(v) \cap (EU \times EU)) \cup R_v$$

$$= (f_1^{\prec}(v) \cap (EU \times EU)) \cup R_v = f_2^{\prec}(v).$$

- $s_2 > k$. Then $s_1 > k$, and hence $s > k$.
- $s_2 \leq k$. Thus $s_1 \leq k$, and hence $s = s_1 = s_2$ and $o = o_1 = o_2$.

Thus $h \in h_2 \downarrow k$ which, by $h_2 \in \mathcal{M}(S)$, implies $h \in \mathcal{M}(S) \downarrow k$.

9.1.3. Proof of Expressibility Connect

The proof of Theorem 9.4 proceeds similar to the proof above. Consider a statechart S. Let, throughout this section, $D_0 = \mathcal{H}$ and $D_{k+1} = CONC(\mathcal{M}(S), D_k, T_1, T_2)$. We prove that there exists an assertion $\xi(p)$ such that, for all $i \in \mathbb{N}$, $\xi[i/p]$ is characteristic for D_i and $FV(\xi) = \{p\}$.

Lemma 9.16. *The sequence $D_i \downarrow k$ for $k = 0, 1, \ldots$, is a nonincreasing sequence (in the subset ordering) of sets that converges to D_i, i.e. $D_i = \bigcap_{k \in \mathbb{N}} D_i \downarrow k$.*

Proof. This lemma follows directly from the proof of Lemma 9.7 (see Section 9.1.1). □

Lemma 9.17. $D_i \downarrow k = \{h \in \mathcal{H} \mid h \downarrow (k, EU) \in D_i \downarrow (k, EU)\} \cup \{h \in \mathcal{H} \mid \hat{s} > k\}$.

Proof. Similar to the proof of Lemma 9.10, since the properties of $\mathcal{M}(S)$ we used there also hold for D_i. □

Definition 9.18. $D_i^c \downarrow (k, EU) = \{code(h \downarrow (k, EU)) \mid h \downarrow (k, EU) \in D_i \downarrow (k, EU)\}$.

Lemma 9.19. $\{(l, k, i) \mid l \in D_i^c \downarrow (k, EU), k \in \mathbb{N}, l \in \mathbb{N}, i \in \mathbb{N}\}$ *is a recursively enumerable set.*

Proof. Since

$$\{(l, k, i) \mid l \in D_i^c \downarrow (k, EU), l \in \mathbb{N}, k \in \mathbb{N}, i \in \mathbb{N}\}$$

$$= \bigcup_{k \in \mathbb{N}} \bigcup_{i \in \mathbb{N}} \{(l, k, i) \mid l \in D_i^c \downarrow (k, EU), l \in \mathbb{N}\},$$

it is sufficient to show that $D_i^c\downarrow(k, EU)$ is r.e. for all $k, i \in \mathbb{N}$. Similar to the proof of Lemma 9.12 this follows from the observation that $D_i\downarrow(k, EU)$ is finite, for all $k, i \in \mathbb{N}$. □

Hence, by recursion theory, we obtain the following lemma.

Lemma 9.20. *There exists an assertion $\psi(n, m, p)$ with $\mathrm{FV}(\psi) = \{n, m, p\}$ such that for all $k, l, i \in \mathbb{N}$, $\models \psi[k/n, l/m, i/p]$ iff $l \in D_i^c\downarrow(k, EU)$.*

Finally we prove Theorem 9.4. Let $i \in \mathbb{N}$. From Lemma 9.20 and Lemma 9.14 we obtain assertions $\psi(n, m, p)$ and $\chi(n, m)$. Define

$$\varphi(n, p) = (\exists m\colon \psi(n, m, p) \wedge \chi(n, m)) \vee (st > p),$$

then $\mathrm{FV}(\varphi) = \{n, p\}$. Now we prove that for $k \in \mathbb{N}$, $\varphi[k/n, i/p]$ characterizes $D_i\downarrow k$, i.e. $[\![\varphi[k/n, i/p]]\!] = D_i\downarrow k$. Let envuronment γ and $h \in \mathscr{H}$ be arbitrary. Then

$[\![\varphi[k/n, i/p]]\!]\gamma h$
iff, by definition of φ,
there exists $l \in \mathbb{N}$ such that $[\![\psi[k/n, l/m, i/p]]\!]\gamma h$ and $[\![\chi[k/n, l/m]]\!]\gamma h$, or $\hat{s} > k$
iff using Lemmas 9.20 and 9.14,
there exists $l \in \mathbb{N}$ such that $l \in D_i^c\downarrow(k, EU)$ and $code(h\downarrow(k, EU)) = l$, or $\hat{s} > k$
iff
$code(h\downarrow(k, EU)) \in D_i^c\downarrow(k, EU)$ or $\hat{s} > k$
iff, by Definition 9.18,
$h\downarrow(k, EU) \in D_i\downarrow(k, EU)$ or $\hat{s} > k$
iff, using Lemma 9.17 (recall that $h \in \mathscr{H}$),
$h \in D_i\downarrow k$.

Now define $\xi(p) \equiv \forall n\colon \varphi(n, p)$. Then $\mathrm{FV}(\xi) = \{p\}$ and $[\![\xi[i/p]]\!] = [\![\forall n\colon \varphi(n)[i/p]]\!] = \bigcap_{k \in \mathbb{N}} [\![\varphi[k/n, i/p]]\!] = \bigcap_{k \in \mathbb{N}} D_i\downarrow k = D_i$, by Lemma 9.16.

9.2. Derivability

Remains to prove Theorem 9.5, that is, for every statechart S we prove $\vdash S$ **sat** φ for an assertion φ which is characteristic for S. We strengthen the theorem and prove by induction on the structure of S that there exists an assertion φ such that

(i) φ is characteristic for S,
(ii) φ does not contain any free logical variables, i.e. $\mathrm{FV}(\varphi) = \emptyset$, and
(iii) $\vdash S$ **sat** φ.

Basic Statechart. Consider

$$\varphi \equiv (in \in I \vee in = \star) \wedge \forall k, st < k < es\colon WAIT(O, k)$$
$$\wedge [(out = \perp \wedge es = \infty) \vee (out = \star \wedge F(es) = \emptyset) \vee FIRE(O, es)].$$

(i) We show that φ is characteristic for $[I, O, N]$. From soundness of the Basic Statechart Axiom we obtain $\mathcal{M}([I, O, N]) \subseteq [\![\varphi]\!]$. Remains to prove:

$$[\![\varphi]\!] \subseteq \mathcal{M}([I, O, N]).$$

Consider $h \in [\![\varphi]\!]$. Let γ be arbitrary. Then $[\![\varphi]\!]\gamma h$, and thus, by definition of φ, $i \in (I \cup \{\star\})$ and, using Lemma 8.3, $\forall v, \hat{s} < v < s$: $wait(O, v)$. Similarly, by Lemma 8.4, we obtain $fire(O, s) \vee (o = \star \wedge f^F(s) = \emptyset) \vee s = \infty$. Thus $h \in \mathcal{M}([I, O, N])$.

(ii) Clearly φ does not contain any free logical variables.

(iii) $\vdash [I, O, N]$ **sat** φ by the Basic Statechart Axiom.

Or. By the induction hypothesis we obtain φ_1 and φ_2 for S_1 and S_2, respectively.

(i) Since φ_1 and φ_2 are characteristic for, respectively, S_1 and S_2 we obtain

$$[\![\varphi_1 \vee \varphi_2]\!] = [\![\varphi_1]\!] \cup [\![\varphi_2]\!] = [\![S_1]\!] \cup [\![S_2]\!] = \mathcal{M}(Or(S_1, S_2)).$$

Hence $\varphi_1 \vee \varphi_2$ is characteristic for $Or(S_1, S_2)$.

(ii) $\mathrm{FV}(\varphi_1 \vee \varphi_2) = \emptyset$, since $\mathrm{FV}(\varphi_1) = \emptyset$ and $\mathrm{FV}(\varphi_2) = \emptyset$.

(iii) Since $\vdash S_1$ **sat** φ_1 and $\vdash S_2$ **sat** φ_2, the Or Rule leads to $\vdash Or(S_1, S_2)$ **sat** $\varphi_1 \vee \varphi_2$.

Connection. By the induction hypothesis we obtain $\hat{\varphi}$ with $\mathrm{FV}(\hat{\varphi}) = \emptyset$ such that $\vdash S$ **sat** $\hat{\varphi}$, with $\hat{\varphi}$ characteristic for S. From the definition of the semantics,

$$\mathcal{M}(Connect(S, T_1, T_2)) = del_{T_2}\left(\bigcap_{k \in \mathbb{N}} D_k\right)$$

with $D_0 = \mathcal{H}$ and $D_{k+1} = CONC(\mathcal{M}(S), D_k, T_1, T_2)$ for $k \in \mathbb{N}$. From Theorem 9.4, there exists an assertion $\varphi(n)$ in our assertion language such that $D_k = [\![\varphi[k/n]]\!]$, for all $k \in \mathbb{N}$.

(i) From soundness of the Connect Rule we obtain

$$\mathcal{M}(Connect(S, T_1, T_2)) \subseteq [\![in \neq T_2 \wedge \forall n\colon \varphi(n)]\!].$$

For the reverse inclusion, let γ and h be arbitrary, and assume $[\![in \neq T_2 \wedge \forall n\colon \varphi(n)]\!]\gamma h$. Then $i \neq T_2$ and $h \in D_k$, for all $k \in \mathbb{N}$. Hence

$$h \in del_{T_2}\left(\bigcap_{k \in \mathbb{N}} D_k\right) = \mathcal{M}(Connect(S, T_1, T_2)).$$

Thus $in \neq T_2 \wedge \forall n\colon \varphi(n)$ is characteristic for $Connect(S, T_1, T_2)$.

(ii) $\mathrm{FV}(in \neq T_2 \wedge \forall n\colon \varphi(n)) = \emptyset$, since $\mathrm{FV}(\varphi(n)) = \{n\}$.

(iii) To apply the Connect Rule, observe

(1) $\models \varphi(0)$, since $[\![\varphi(0)]\!] = D_0 = \mathcal{H}$. Hence, by the relative completeness assumption, $\varphi(0)$ is derivable.

(2) We prove $\models conc(\hat{\varphi}, \varphi(n), T_1, T_2) \rightarrow \varphi[n+1/n]$, that is,

$$\begin{aligned} \models\ & (\hat{\varphi} \wedge out \neq T_1) \\ & \vee (\hat{\varphi}[f_1/F, T_1/out, v/es] \wedge \varphi(n)[v/st, T_2/in, f_2/F] \wedge F = f_1 \cup f_2) \\ & \rightarrow\ \varphi[n+1/n], \end{aligned}$$

where v, f_1, f_2 are fresh logical variables. Let γ and h be arbitrary. Suppose $\gamma(n) = k$ for some constant $k \in \mathbb{N}$. Assume

$$[\![(\hat{\varphi} \wedge out \neq T_1) \vee (\hat{\varphi}[f_1/F, T_1/out, m/es] \wedge \varphi(n)[m/st, T_2/in, f_2/F] \wedge F = f_1 \cup f_2)]\!]\gamma h.$$

Then there are two possibilities.

(a) $[\![\hat{\varphi} \wedge out \neq T_1]\!]\gamma h$. Since $FV(\hat{\varphi}) = \emptyset$, we have $[\![\hat{\varphi}]\!]\gamma h$, for all γ, and thus $h \in [\![\hat{\varphi}]\!] = \mathcal{M}(S)$. Furthermore, $o \neq T_1$, and hence

$$h \in \{h \mid h \in \mathcal{M}(S) \wedge o \neq T_1\} \subseteq CONC(\mathcal{M}(S), D_k, T_1, T_2).$$

Thus $h \in D_{k+1} = [\![\varphi[k+1/n]]\!]$. Since $\gamma(n) = k$ this leads to $[\![\varphi[n+1/n]]\!]\gamma h$.

(b) $[\![\hat{\varphi}[f_1/F, T_1/out, m/es] \wedge \varphi(n)[m/st, T_2/in, f_2/F] \wedge F = f_1 \cup f_2]\!]\gamma h$. Then $f^F = \gamma(f_1) \cup \gamma(f_2)$. Define

$$h_1 = (\hat{s}, i, \langle \gamma(f_1), f^C, f^{\prec} \rangle, T_1, \gamma(m))$$

and

$$h_2 = (\gamma(m), T_2, \langle \gamma(f_2), f^C, f^{\prec} \rangle, o, s).$$

Then from $[\![\hat{\varphi}[f_1/F, T_1/out, m/es]]\!]\gamma h$ and $[\![\varphi(n)[m/st, T_2/in, f_2/F]]\!]\gamma h$ we obtain $[\![\hat{\varphi}]\!]\gamma h_1$ and $[\![\varphi(n)]\!]\gamma h_2$. Hence $[\![\varphi[k/n]]\!]\gamma h_2$, because $\gamma(n) = k$. Since $FV(\hat{\varphi}) = FV(\varphi[k/n]) = \emptyset$, we obtain $h_1 \in [\![\hat{\varphi}]\!]$ and $h_2 \in [\![\varphi[k/n]]\!]$. Thus $h_1 \in \mathcal{M}(S)$ and $h_2 \in D_k$. By definition of h_1 and h_2 this leads to $h \in D_{k+1}$, and hence $[\![\varphi[k+1/n]]\!]\gamma h$, that is, $[\![\varphi[n+1/n]]\!]\gamma h$. This proves $\models conc(\varphi, \varphi(n), T_1, T_2) \rightarrow \varphi[n+1/n]$ and thus, by the relative completeness assumption, $conc(\varphi, \varphi(n), T_1, T_2) \rightarrow \varphi[n+1/n]$ is derivable.

By the induction hypothesis we obtain $\vdash S$ **sat** $\hat{\varphi}$. Since $FV(\hat{\varphi}) = \emptyset$, n does not occur free in $\hat{\varphi}$. Hence the Connect Rule leads to $Connect(S, T_1, T_2)$ **sat** $in \neq T_2 \wedge \forall n: \varphi(n)$.

And. By the induction hypothesis we obtain φ_1 and φ_2 such that $\vdash S_1$ **sat** φ_1 and $\vdash S_2$ **sat** φ_2, with φ_1 and φ_2 characteristic for S_1 and S_2, respectively. By Theorem 9.3 there exists an assertion φ such that

(i) φ is characteristic for $And(S_1, S_2)$, and

(ii) $FV(\varphi) = \emptyset$.

(iii) Remains to prove $\vdash And(S_1, S_2)$ **sat** φ. To apply the And Rule, we have to show

$$\models \varphi_1[t_1^i/in, f_1/F, t_1^o/out] \wedge \varphi_2[t_2^i/in, f_2/F, t_2^o/out] \wedge and\_in \wedge and\_out \wedge F = f_1 \cup f_2 \rightarrow \varphi.$$

(By the relative completeness assumption this implication is then also derivable.) Let γ and h be such that $[\![\varphi_1[t_1^i/in, f_1/F, t_1^o/out]]\!]\gamma h$, $[\![\varphi_2[t_2^i/in, f_2/F, t_2^o/out]]\!]\gamma h$, $[\![and\_in \wedge and\_out]\!]\gamma h$ and $[\![F = f_1 \cup f_2]\!]\gamma h$. Define

$$h_1 = (\hat{s}, \gamma(t_1^i), \langle \gamma(f_1), f^C, f^{\prec} \rangle, \gamma(t_2^o), s)$$

and

$$h_2 = (\hat{s},\ \gamma(t_2^i),\ \langle \gamma(f_2), f^C, f^{\prec}\rangle,\ \gamma(t_2^o),\ s).$$

Then $[\![\varphi_1]\!]\gamma h_1$ and $[\![\varphi_2]\!]\gamma h_2$. From $FV(\varphi_1) = FV(\varphi_2) = \emptyset$ we obtain $h_1 \in [\![\varphi_1]\!]$ and $h_2 \in [\![\varphi_2]\!]$. Thus $h_1 \in \mathcal{M}(S_1)$ and $h_2 \in \mathcal{M}(S_2)$. Since $i_1 \equiv \gamma(t_1^i)$, $o_1 \equiv \gamma(t_2^o)$, $i_2 \equiv \gamma(t_2^i)$, $o_2 \equiv \gamma(t_2^o)$ and $f^F = \gamma(f_1) \cup \gamma(f_2)$, we obtain from $[\![and\_in \wedge and\_out]\!]\gamma h$ that

$$(i = i_1 = i_2) \vee (i = i_1 \wedge i_2 = \star) \vee (i = i_2 \wedge i_1 = \star) \quad \text{and}$$

$$(o = o_1 \neq \perp \wedge o_2 = \star) \vee (o = o_2 \neq \perp \wedge o_1 = \star) \vee (o = o_1 = o_2 = \perp).$$

Consequently $h \in \mathcal{M}(And(S_1, S_2))$ and hence, using that φ is characteristic for $And(S_1, S_2)$, $[\![\varphi]\!]\gamma h$.

Statification. The proof for $Stat(B, S, T)$ is similar to the proof for the And-construct given above. The only difference is the way of entering the construct.

Hide-Closure. By the induction hypothesis we obtain an assertion φ such that $FV(\varphi) = \emptyset$, $\vdash S$ **sat** φ and φ is characteristic for S. By Theorem 9.3 there exists an assertion φ' such that

(i) φ' is characteristic for $HiCl(S, a)$, and

(ii) $FV(\varphi') = \emptyset$.

(iii) Remains to prove $\vdash HiCl(S, a)$ **sat** φ'. From Theorem 9.3 we obtain that a does not occur in φ', since $a \notin Events(HiCl(S, a))$. In order to apply the Hide-Closure Rule we have to prove

$$\vdash \varphi \wedge (\forall v\colon occ(a, v) \to a \in F(v)) \to \varphi'[F \dot{-} \{a\}/F].$$

Let γ and h be arbitrary. Assume $[\![\varphi]\!]\gamma h$ and $[\![\forall n\colon occ(a, n) \to a \in F(n)]\!]\gamma h$. Since $FV(\varphi) = \emptyset$, we obtain $h \in [\![\varphi]\!]$ and thus $h \in \mathcal{M}(S)$. Furthermore, $\forall v\colon a \in f^C(v) \to a \in f^F(v)$. Let h_1 such that $\hat{s}_1 = \hat{s}$, $i_1 = i$, $o_1 = o$, $s_1 = s$, $f_1^F = f^F \dot{-} \{a\}$, $f_1^C \dot{-} \{a\} = f^C \dot{-} \{a\}$ and $f_1^{\prec}|_a = f^{\prec}|_a$. Then $h_1 \in \mathcal{M}(HiCl(s, a))$, thus $[\![\varphi']\!]\gamma h_1$, and hence

$$[\![\varphi'[F \dot{-} \{a\}/F]]\!]\gamma h.$$

10. Conclusion and future work

The present paper offers a compositional axiomatic system for both safety and liveness properties which is as simple as we can imagine for Statecharts. As a consequence, one can object that we merely axiomatize the semantics. The techniques given in [17, 18, 22, 26, 34, 35, 36], however, illustrate that such a simple formalism can be lifted to a more sophisticated formalism, e.g. based on assumption-commitment or rely-guarantee pairs. Determining which formalism is most elegant for Statecharts is left to future research. Other future work in the line of, e.g. [28, 34],

is an extension of the axiomatic system to a mixed specification formalism in which systems are described partly behaviourally and partly in terms of their properties.

One of the main aims in the present work has been to investigate how the elegant compositional verification techniques for safety properties from Zwiers [34] could be extended to liveness properties. In discussing this extension with Amir Pnueli, the issue has been raised as to the exact relationship between proving liveness properties by means of higher ordinals, and proving liveness on the basis of natural numbers (using that these are able to code the precise computations). This becomes especially interesting when we add assignments to variables and boolean tests on these variables to the language. We illustrate this topic in the context of Statecharts by an example.

Example 10.1. We extend our version of Statecharts with program variables and, as in the full language [11], labels are of the form $E[b]/A$ where $[b]$ is the *condition* part, consisting of a boolean expression b in terms of the program variables. A transition with such a label can only be taken if the condition part evaluates to true. The set of actions A is extended with assignments to program variables. For instance, a transition with label $(a \wedge b)[x = 5]/\{c, y := x+1\}$ can be taken if the events a and b occur and x has the value 5. When this transition is taken event c is generated and the assignment $y := x+1$ is executed. In the example below we use a label $[x > 0]/x := x-1$ as an abbreviation of $\lambda[x > 0]/\{x := x-1\}$, and a label $[x = 0]$ as an abbreviation of $\lambda[x = 0]/\emptyset$.

We demonstrate that our rule for the Connect-construct can also be used to derive liveness properties for this extended version of Statecharts. Assume the assertion language includes $x(n)$ to denote the value of variable x at the end of step n (and thus at the start of step $n+1$).

Consider statechart $S \equiv [\{T_0, T_2\}, \{(T_3, [x = 0]), (T_1, [x > 0]/x := x-1)\}, A]$ and $Connect(S, T_1, T_2)$ (see, resp., Fig. 8 and Fig. 9).

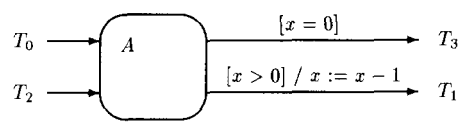

Fig. 8. Statechart S.

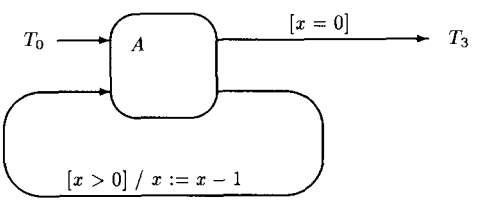

Fig. 9. $Connect(S, T_1, T_2)$.

We prove $Connect(S, T_1, T_2)$ **sat** $x(st) \geq 0 \rightarrow (out = T_3 \wedge es = st+1+x(st))$. By an obvious extension of the axiom for basic statecharts we can easily derive S **sat** $\hat{\varphi}$, with

$$\hat{\varphi} \equiv x(st) \geq 0 \rightarrow (es = st+1 \wedge [(x(st) = 0 \wedge out = T_3) \vee (x(st) > 0 \wedge x(st+1) = x(st) - 1 \wedge out = T_1)]).$$

Define $\varphi(n) \equiv (n > 0 \wedge 0 \leq x(st) < n) \rightarrow (out = T_3 \wedge es = st+1+x(st))$.

Clearly, $\varphi(0)$ holds. Assume $conc(\hat{\varphi}, \varphi(n), T_1, T_2)$, which is equivalent to $(\hat{\varphi} \wedge out \neq T_1) \vee (\hat{\varphi}[T_1/out, m/es] \wedge \varphi(n)[m/st, T_2/in])$. Thus

(1) $x(st) \geq 0 \rightarrow (es = st+1 \wedge x(st) = 0 \wedge out = T_3)$, or

(2) $[x(st) \geq 0 \rightarrow (m = st+1 \wedge x(st) > 0 \wedge x(st+1) = x(st) - 1)]$
$\wedge [(n > 0 \wedge 0 \leq x(m) < n) \rightarrow (out = T_3 \wedge es = m+1+x(m))]$.

We have to prove $\varphi(n+1)$, i.e. $(0 \leq x(st) < n+1) \rightarrow (out = T_3 \wedge es = st+1+x(st))$. Assume $0 \leq x(st) < n+1$.

If (1) above holds, then $es = st+1 \wedge x(st) = 0 \wedge out = T_3$, and thus

$$out = T_3 \wedge es = st+1+x(st).$$

If (2) holds then, using $x(st) \geq 0$, we obtain $x(st) > 0$. Furthermore, observe that if $n = 0$ then $0 \leq x(st) < n+1$ leads to $0 \leq x(st) < 1$, and thus $x(st) = 0$. Hence $n > 0$. From $0 \leq x(st) < n+1$ and $x(st) > 0$ we obtain $0 \leq x(st) - 1 < n$. By $m = st+1 \wedge x(st+1) = x(st) - 1$, this leads to $0 \leq x(m) < n$. Then, from (2), $out = T_3 \wedge es = m+1+x(m)$. Thus using $m = st+1$, we obtain $out = T_3 \wedge es = st+1+1+x(st+1)$, and hence, by $x(st+1) = x(st) - 1$, $out = T_3 \wedge es = st+1+x(st)$. This proves $\varphi(n+1)$. Then the connect rule leads to $\forall n$: $\varphi(n)$ for $Connect(S, T_1, T_2)$. Remains to prove that $\forall n$: $\varphi(n)$ implies $x(st) \geq 0 \rightarrow (out = T_3 \wedge es = st+1+x(st))$. Therefore, assume $x(st) \geq 0$, and let k be such that $k > x(st)$. From $\forall n$: $\varphi(n)$, we obtain $\varphi[k/n]$, and thus, by definition of $\varphi(n)$, this leads to

$$out = T_3 \wedge es = st+1+x(st).$$

In this example we have proved liveness properties of a statechart by first expressing the behaviour of the statechart up to step n after the entry step, and then quantifying over all n. Current work includes the application of these ideas to the specification and compositional verification of real-time properties of Occam-like programs.

Acknowledgement

Our thanks goes to the members of AdCAD, the Weizmann Institute of Science, an the Eindhoven University of Technology which have been associated with Esprit project DESCARTES. Their comments on the work described here and their participation in extensive discussions about the semantics of Statecharts are gratefully

acknowledged. We would like to thank the referees for their valuable suggestions which have led to numerous improvements.

References

[1] H. Barringer, R. Kuiper and A. Pnueli, Now you may compose temporal logic specifications, in: *Proc. 16th Ann. ACM Symp. on Theory of Computing* (1984) 51-63.

[2] B. Berry and L. Cosserat, The synchronous programming language Esterel and its mathematical semantics, in: *Proc. CMU Seminar on Concurrency*, Lecture Notes in Computer Science, Vol. 197 (Springer, Berlin, 1985) 389-449.

[3] P. Caspi, D. Pilaud, N. Halbwachs and J. Plaice, Lustre: a declarative language for programming synchronous systems, in: *Proc. 14th ACM Symp. on Principles of Programming Languages* (1987) 178-188.

[4] J.W. de Bakker, *Mathematical Theory of Program Correctness* (Prentice-Hall, Englewood Cliffs, NJ, 1980).

[5] D. Drusinsky and D. Harel, On the power of cooperative concurrency, in: *Proc. Concurrency '88*, Lecture Notes in Computer Science, Vol. 335 (Springer, Berlin, 1988) 74-103.

[6] D. Drusinsky and D. Harel, Using statecharts for hardware description and synthesis, *IEEE Trans. Computer-Aided Design* **8** (1989) 798-807.

[7] N. Francez, *Fairness* (Springer, Berlin, 1986).

[8] G. Gonthier, Sémantiques et modèles d'exécution des langages réactifs synchrones; Application à ESTEREL, Ph.D. thesis, University of Orsay, Paris, 1988.

[9] P. le Guernic and A. Benveniste, Real-time, synchronous, data-flow programming: The language Signal and its mathematical semantics, Technical Report 620, INRIA, Rennes, 1986.

[10] D. Harel, *First-Order Dynamic Logic*, Lecture Notes in Computer Science, Vol. 68 (Springer, Berlin, 1979).

[11] D. Harel, Statecharts: A visual formalism for complex systems, *Sci. Comput. Programming* **8**(3) (1987) 231-274.

[12] D. Harel, On visual formalisms, *Comm. ACM* **31** (1988) 514-530.

[13] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring and M. Trakhtenbrot, STATEMATE: A working environment for the development of complex reactive systems, *IEEE Trans. Software Eng.* **16** (1990) 403-414.

[14] D. Harel and A. Pnueli, On the development of reactive systems, in: *Logics and Models of Concurrent Systems*, NATO, ASI-13 (Springer, Berlin, 1985) 477-498.

[15] D. Harel, A. Pnueli, J. Pruzan-Schmidt and R. Sherman, On the formal semantics of Statecharts, in: *Proc. Symp. on Logic in Computer Science* (1987) 54-64.

[16] C.A.R. Hoare, *Communicating Sequential Processes* (Prentice-Hall, Englewood Cliffs, NJ, 1985).

[17] J. Hooman, A compositional proof theory for real-time distributed message passing, in: *Parallel Architectures and Languages Europe*, Lecture Notes in Computer Science, Vol. 259 (Springer, Berlin, 1987) 315-332.

[18] J. Hooman, Compositional verification of distributed real-time systems, in: *Proc. Workshop on Real-Time Systems—Theory and Applications* (North-Holland, Amsterdam, 1990) 1-20.

[19] J. Hooman and J. Widom, A temporal-logic-based compositional proof system for real-time message passing, in: *Parallel Architectures and Language Europe, Vol. II*, Lecture Notes in Computer Science, Vol. 366 (Springer, Berlin, 1989) 424-441.

[20] C. Huizing, R. Gerth and W.P. de Roever, Modelling statecharts behaviour in a fully abstract way, in: *Proc. 13th Coll. on Trees in Algebra and Programming*, Lecture Notes in Computer Science, Vol. 299 (Springer, Berlin, 1988) 271-294.

[21] C. Huizing and W.P. de Roever, Introduction to design choices in the semantics of statecharts, *Inform. Processing Lett.* **37** (1991) 205-213.

[22] C.B. Jones, Tentative steps towards a development method for interfering programs, *ACM Trans. Prog. Lang. Sys.* **5**(4) (1983) 596-619.

[23] R. Koymans, R.K. Shyamasundar, W.P. de Roever, R. Gerth and S. Arun-Kumar, Compositional semantics for real-time distributed computing, *Inform. and Comput.* **79**(3) (1988) 210–256.

[24] L. Lamport, What Good is Temporal Logic, in: R.E. Manson, ed., *Information Processing* (North-Holland, Amsterdam, 1983) 657–668.

[25] F. Maraninchi, Argonaute: graphical description, semantics and verification of reactive systems by using a process algebra, in: *Proc. Workshop on Automatic Verification Methods for Finite State Systems*, Lecture Notes in Computer Science, Vol. 407 (Springer, Berlin, 1989) 38–53.

[26] J. Misra and K.M. Chandy, Proofs of networks of processes, *IEEE Trans. Software Eng.* **7**(7) (1981) 417–426.

[27] V. Nguyen, A. Demers, D. Gries and S. Owicki, A model and temporal proof system for networks of processes, *Distrib. Comput.* **1**(1) (1986) 7–25.

[28] E.R. Olderog, Process theory: semantics, specification and verification, in: *ESPRIT/LPC Advanced School on Current Trends in Concurrency*, Lecture Notes in Computer Science, Vol. 194 (Springer, Berlin, 1985) 509–519.

[29] P. Place, W. Wood and M. Tudball, Survey of formal specification techniques for reactive systems, Technical Report CMU/SEI-90-TR-5, Software Engineering Institute, Carnegie-Mellon University, 1990.

[30] A. Pnueli and M. Shalev, What is in a step?, Technical report, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, Israel, 1988.

[31] A.U. Shankar and S.S. Lam, Time-dependent distributed systems: proving safety, liveness and real-time properties, *Distrib. Comput.* **2** (1987) 61–79

[32] J. Sifakis, ed., *Proc. Workshop on Automatic Verification Methods for Finite State Systems* (Springer, Berlin, 1989).

[33] K. Weihrauch, *Computability*, EATCS Monographs on Theoretical Computer Science, Vol. 9 (Springer, Berlin, 1987).

[34] J. Zwiers, *Compositionality, Concurrency and Partial Correctness*, Lecture Notes in Computer Science, Vol. 321 (Springer, Berlin, 1989).

[35] J. Zwiers and W.P. de Roever, Predicates are predicate transformers: a unified compositional theory for concurrency, in: *Proc. 8th ACM Symp. on Principles of Distributed Computing* (1989).

[36] J. Zwiers, W.P. de Roever and P. van Emde Boas, Compositionality and concurrent networks: soundness and completeness of a proof system, Technical Report 57, University of Nijmegen, The Netherlands, 1984.